Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Blockchained Spreadsheets

Marvin Tiedtke

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Bernhard Mitschang
Betreuer/in:	Dennis Przytarski, M.Sc.
Beginn am:	15. Mai 2019
Beendet am:	15. November 2019

Kurzfassung

Tabellen werden in vielen Firmen verwendet. Insbesondere im Rechnungswesen haben sie eine große Bedeutung und sind häufig Entscheidungshilfe zur zukünftigen Firmenausrichtung. Die Fehlerfreiheit zu garantieren ist schwierig, da neben den normalen Nutzungsfehlern, auch häufig mehrere Mitarbeiter Zugriff auf dasselbe Dokument haben, woraus unterschiedliche Versionen entstehen können. Um diesen Fehlerquellen entgegenzuwirken, können Änderungsprotokolle angefertigt werden. Diese erlauben die Durchführung von Audit Trails, eine Evaluierung in Tabellen gehaltenen Informationen auf ihre Korrektheit. Allerdings schützt dies Firmen nicht vor gezielter Manipulation, da Änderungsprotokolle sowohl während der Erstellung, als auch nachträglich verfälscht werden können. Die Blockchain, mit einigen der ihr zugrunde liegenden Eigenschaften wie Unveränderbarkeit, Konsistenz und Manipulationsresistenz, könnte eine permanente Speichermöglichkeit für diese Änderungsprotokolle darstellen. Smart Contracts erlauben eine Ausführung von selbst entwickelten Programmen unter Nutzung der Blockchain-Technologie. Dadurch können für dieses Programm die gleichen Eigenschaften wie für die Blockchain selbst garantiert werden. Das Speichern von Datensätzen zählt jedoch zu den teureren Operationen in Smart Contracts. In diesem Kontext mögliche Umsetzungen von manipulationsresistenten Tabellenänderungsprotokollen und deren verbundene Kosten untersucht. Zu diesem Zweck werden mehrere Speicherstrukturen für Tabellenänderungsprotokolle als Smart Contracts umgesetzt. Eine zusätzliche Webkomponente, um vergangener Änderungen des vermutlich kosteneffizientesten Ansatzes zu visualisieren, wird entwickelt. Darauf wird bestehende Tabellensoftware auf ihre Umsetzungskompatibilität bewertet und die zuvor entwickelten Smart Contracts in den Arbeitsfluss des ausgewählten LibreOffice Calcs integriert. Alle Änderungen werden zwischengespeichert bis der Nutzer eine Externalisierung anstößt. Die durchgeführten Änderungen werden darauf auf eine Ethereum-Blockchain geschrieben. Die durchgeführte Kostenabschätzungen hat die vorhergesagten Kostenverteilungen ungefähr bestätigt. Das Spiegeln des Ist-Zustands der kompletten Tabelle ist am teuersten und wird eine geringe Etablierungchance in der Realität zugeschrieben. Das Speichern der letzten Änderungen in den Ethereum-Event-Logs, encodiert in einem etablierten Serialisierungsformat, kann Kosten dieses Konzepts signifikant verringern. Jeder fortlaufende Monat des Vergleichssystems erlaubt die Speicherung von 15 Änderungen mit den Blockchain-Eigenschaften ohne Mehrkosten. Damit könnte dieser Ansatz für ausgewählte Tabellen mit hoher Entscheidungsrelevanz in Betracht gezogen werden. Abschließend wurde dieser Ansatz um Methoden zur Beschränkung der Änderungserlaubnis erweitert.

Abstract

Spreadsheets are used in many businesses amongst others for financial record keeping. Maintaining correctness is difficult, given that besides normal user errors, multiple employees can have access to the same document, resulting in multiple versions. Creating change logs and often performing audit trails, i.e. reevaluating spreadsheets, can counteract some error sources. Faults emerging from malicious or fraudulent behavior, either during log creation or after the fact, unfortunately can hardly be detected by this method. The blockchain could be used to permanently store these change logs, given some of its underlying properties are immutability, consistency, and tamper resistance. Everyone can develop programs, which can be executed using the blockchain technology and thus sharing the underlying properties. These programs are called smart contracts. Unfortunately is storing data on the blockchain considered one of the more costly use cases. In this context we research the possibility of storing tamper resistant spreadsheet change logs. This work aims to develop multiple realizations of spreadsheet change logs on the Ethereum blockchain and explore its associated costs. The presumed cheapest implementation supports a web application to visualize recorded modifications. Existing spreadsheet applications are analyzed to find the best possible fit to integrate these developed smart contracts. Changes performed in LibreOffice Calc are buffered until a user starts an exporting operation. These are then stored on-chain. The hypothesized cost distribution of developed smart contracts were found to be roughly accurate. Maintaining the current table state was found to be the most expensive approach and its chances of success in the real world are considered slim. Only storing the most recent changes, serialized in a well-established format, in the Ethereum event log was found to significant reduce costs of this concept. Each month the data would be stored in a traditional system, is equivalent to 15 changes with our endorsed solution. This limits possible real world application to a select few key files, holding decision power in businesses. Lastly we extended this implementation with methods restricting editing rights.

Inhaltsverzeichnis

1 Einleitung **11**

2 Related Work **13**
2.1 Spreadsheets . . . 13
2.2 Blockchain . . . 16
2.2.1 Ethereum . . . 17
2.2.2 Anwendungsbereiche . . . 23

3 Konzept **25**
3.1 Entwurfsziele . . . 25
3.2 Blockchain-Transaktionsauslösung . . . 26
3.3 Audit Trail Aufbau . . . 27
3.3.1 Metadaten . . . 27
3.3.2 Änderungen . . . 28
3.4 Einschränkungen . . . 28
3.5 Zusammenfassung . . . 29

4 Smart Contract Umsetzungen eines Tabellenänderungsprotokolls **31**
4.1 Variante 1: Gespaltene Transaktionen (Split Transactions) . . . 31
4.2 Variante 2: Globale Kopie (Global Copy) . . . 32
4.3 Variante 3: Massentransaktion (Bulk Transaction) . . . 33
4.4 Optimierte Speichermöglichkeiten für Variante 3 . . . 34
4.4.1 Teilweise Encodiert . . . 34
4.4.2 Vollständig Encodiert . . . 36
4.5 Variante 4: Ethereum-Event (Bulk Event Transaction) . . . 36
4.6 Diskussion und Zusammenfassung . . . 36

5 Vergleich von existierenden Tabellenanwendungen **39**
5.1 Bewertungskriterien . . . 39
5.2 Microsoft Excel . . . 40
5.3 LibreOffice Calc . . . 40
5.4 Google Sheets . . . 43
5.5 Diskussion und Auswahl . . . 44
5.6 Zusammenfassung . . . 45

6 Implementierung **47**
6.1 Integration in LibreOffice Calc . . . 47
6.2 Änderungsereignisse . . . 47
6.3 Transaktion . . . 49
6.4 Visualisierung . . . 51

6.5 Zusammenfassung 52

7 Kostenabschätzung **55**
7.1 Vorgehen 55
7.2 Ergebnis 57
7.3 Zusammenfassung 58

8 Diskussion **61**
8.1 Ergebnis 61
8.1.1 Variante 1: Gespaltene Transaktionen (Split Transactions) 62
8.1.2 Variante 2: Globale Kopie (Global Copy) 62
8.1.3 Variante 3: Massentransaktion (Bulk Transaction) 64
8.1.4 Variante 4: Ethereum-Event (Bulk Event Transaction) 64
8.1.5 Vergleich 65
8.2 Einschränkungen 66
8.2.1 Ansatz 66
8.2.2 Implementierung 67
8.3 Folgerungen 69
8.4 Zusammenfassung 70

9 Zusammenfassung und Ausblick **71**

Literaturverzeichnis **75**

Abbildungsverzeichnis

2.1 Benutzeroberfläche des Änderungsprotokolls von Nash et al. [23] 15
2.2 Preisentwicklung der Ethereum-Einheiten von Juni 2019 bis September 2019 . . 18
2.3 Ethereum-Übergänge und deren Anstoß 20
2.4 Vereinfachter Aufbau der Blockchain 24

3.1 Konzeptionelle Komponenten eines Blockchain-basierten Änderungsprotokolls . 30

5.1 *LibreOffice Calc* Ereignisse mit UI-Verbindungsmöglichkeiten für Makros 41
5.2 Natives Änderungsprotokoll von *LibreOffice Calc* 43
5.3 Erweiterungsmöglichkeiten von *LibreOffice Calc* und *Google Sheets* 43

6.1 Grafische Elemente der LibreOffice Calc Erweiterung 48
6.2 Audit Trail Modul Resultate 50
6.3 World State Repräsentation der unoptimierten Massentransaktion nach durchgeführten Änderungen 51
6.4 Weboberfläche des Audit Trails 52
6.5 Technische Umsetzung des Konzepts 53

7.1 Gemessene *Gas*-Kosten nach Smart Contract 56
7.2 *Gas*-Kosten jeder Umsetzung nach Anzahl enthaltener Änderungen 59
7.3 *Gas*-Kosten jeder Umsetzung nach Anzahl in der vorherigen Transaktion enthaltenen Änderungen 60

8.1 *Gas*-Kosten in Abhängigkeit der Anzahl neuer und alter Änderungen 63
8.2 *Gas*-Kosten in Wei pro neuer Änderung der Umsetzungen von Variante 1 64

Tabellenverzeichnis

2.1 *Gas*-Kosten ausgewählter Operationen, vollständige Tabelle in [33] 22

4.1 Speichergrößen der Variante 3 Versionen 37

5.1 Tabellenanwendungsvergleich 46

7.1 *Gas*-Kosten und ihr Wert in US$ des Deployments 56
7.2 Durchschnittliche *Gas*-Kosten aller Externalisierungsevents und pro enthaltener Änderung 57

8.1 *Gas*-Differenz zwischen unterem bzw. oberen Quartil und gemessenen Minimum bzw. Maximum 67

Verzeichnis der Listings

2.1 Einführungsbeispiel von Solidity[1] 21
2.2 Teil der aus Listing 2.1 erstellten OPCodes 21
4.1 Attribute der beiden Umsetzungen von Variante 1 32
4.2 Attribute der Umsetzung von Variante 2 33
4.3 Attribute der Umsetzung von Variante 3 34
4.4 Attribute der teilweise encodieren Optimierung von Variante 3 34
4.5 Minimiertes ProtoBuf Format für Variante 3 35
4.6 Attribute der Event-Optimierung von Variante 3 36
8.1 Event Audit Trail mit unterstützter Versionierung und kontrollierten Editoren . . 69
8.2 Fortgeführte Optimierung des Schemas aus Listing 4.5 auf Seite 35 70

Akronyme

API Application Programming Interface (Programmierschnittstelle). 40, 42, 44, 72

AWS Amazon Web Services. 61, 66

ETH Ether. 17, 18, 19, 20, 58, 66, 67

EuSpRIG European Spreadsheet Risk Interest Group. 11

EVM Ethereum Virtual Machine. 19, 20, 21, 22, 23, 30, 31, 33, 36, 49, 53

IoT Internet of Things (Internet der Dinge). 23

JSON JavaScript Object Notation. 35, 64

ML Machine Learning. 13, 65

PoS Proof-of-Stake. 16

PoW Proof-of-Work. 16

RE Runtime Environment (Laufzeitumgebung). 20

REST Representational State Transfer. 44

SD Standardabweichung. 19, 57

SOX Sarbanes-Oxley Act. 13

SPOF Single Point of Failure (einzelner Ausfallpunkt). 16

UI User Interface (Benutzeroberfläche). 14

UUID Universally Unique Identifier (Universell eindeutiger Identifikator). 47, 52, 72

VBA Visual Basic For Applications. 40

1 Einleitung

Tabellen sind aus dem privaten, aber allem voran auch dem geschäftlichen, Alltag nicht mehr wegzudenken. Häufig werden finanzielle Entscheidungen aufgrund von Daten, die in Spreadsheets gehalten werden, getroffen. Tabellenanwendungen bieten einen generellen Lösungsansatz, der durch guten Tool-Support beliebig erweitert werden kann. Die Verwendung als missionskritische Firmenanwendungen ist oft die Folge [15]. Aufgrund dieses hohen Stellenwertes, überrascht es daher nicht, dass Tabellendokumente immer komplexer werden und ihr Entwicklungsaufwand mit Softwareprojekten vergleichbar sein kann. Im Gegensatz zu dieser fehlen jedoch ausgereifte Methoden der Änderungs-, Daten- und Versionskontrolle [2]. In der Realität enthalten bis zu 50% aller Tabellen Fehler [29]. Diese sind jedoch nicht nur auf menschliches Versagen zurückzuführen. So listen Croll [11] einige Missbräuche mit hohen finanziellen Schäden durch beabsichtigte Manipulation auf. Auch die European Spreadsheet Risk Interest Group (EuSpRIG) listet dutzende Fälle auf, in denen Fehler oder Manipulationen von Spreadsheet-Dokumenten zu monetären oder Imageschäden führten. Diese Skandale, zurückzuführen sowohl auf unbeabsichtigte Fehler, Vernachlässigung von bestehenden Standards, aber auch mutwillige Manipulation, haben die fehlende Zuverlässigkeit einer breiteren Gesellschaftsschicht aufgezeigt und Methoden zur besseren Analyse der enthaltenen Daten hervorgehoben [23]. Um das Fehlerrisiko von Tabellen zu verringern sind Unternehmen zu Risikominimierungsverfahren wie Audit Trails verpflichtet. Jedoch können diese auch nicht jedes Restrisiko eliminieren. So hat sich der damalige Vizepräsident der Finanzen der HealthSouth Corp. zusammen mit mehreren Managern schuldig bekannt, ein falschen Spreadsheet für Auditoren vorbereitet zu haben. Die berichteten Unternehmenseinnahmen wurden auf diese Weise um mindestens 3,5 Milliarden US$ fälschlich erhöht [19].

Die Blockchain-Technologie ist vor einigen Jahren, unter anderem wegen ihrer Eigenschaften, wie Konsistenz und Manipulations- bzw. Fälschungssicherheit, vor allem im Finanzsektor populär geworden. Sie dringt allerdings auch in andere Anwendungsgebiete ein. Darüber hinaus erlaubte die Einführung von Smart Contracts in Ethereum die Umsetzung neuer Arten von Applikationen. Ein Einsatz der Blockchain als Datenspeicher ist denkbar und Teil aktueller Forschungsarbeiten. So ist es möglich, Veränderungen von Untersuchungsergebnissen über die Blockchain mit konstanten Kosten zu verfolgen [30].

Eine Speicherung von Änderungsprotokollen auf der Blockchain kann sich als eine effiziente und sichere Umsetzung von Audit Trails erweisen. Daten werden mit einmaligen Kosten permanent auf der Blockchain hinterlegt und eine nachträgliche Manipulation kann nahezu ausgeschlossen werden. Obwohl durch diesen Ansatz vergangene Aktivitäten Schritt für Schritt nachvollzogen werden können, konnte keine Umsetzung oder Untersuchung gefunden werden.

Im Weiteren soll in dieser Arbeit die Machbarkeit und Effizienz von Tabellenänderungsprotokollen auf der Blockchain untersucht werden. Dazu werden, nach der Konkretisierung des Konzepts, mehrere potenzielle Umsetzungsvarianten eines Änderungsprotokolls auf der Ethereum-Blockchain vorgestellt. Darauf folgt ein Vergleich von existierenden Tabellenanwendungen, um die informierte

Auswahl eines Programms zu ermöglichen. Dieses wird insoweit erweitert, dass es die Aufzeichnung von Änderungen und anschließende Weiterleitung an die Ethereum-Blockchain unterstützt. Diese Änderungen können darauf durch eine zusätzlich entwickelte Webkomponente visualisiert werden. Im nächsten Schritt werden die vorgeschlagenen Änderungslogs auf ihre Kosten untersucht. Anhand dieser Kostenabschätzung erfolgt eine Beurteilung der Ergebnisse auf den erreichten Mehrwert und ihre Wirtschaftlichkeit.

Struktur

Nachstehend ist die Struktur dieser Arbeit aufgeführt:

Kapitel 2 – Related Work: In diesem Kapitel werden die zugrunde liegenden Technologien und ihre relevanten Eigenschaften beschrieben. Eine prägnante Übersicht über verwandte akademische Arbeiten ist ebenso Bestandteil dieses Kapitels. Unter anderem werden Relevanz, Schwachstellen und Gegenmaßnahmen von Tabellen sowie die Funktionsweise zusammen mit den Vor- und Nachteilen der Ethereum-Blockchain aufgeführt.

Kapitel 3 – Konzept: Die zugrunde liegende Idee mitsamt entwickeltem Konzept und die Aufteilung in Teilkomponenten wird hier erläutert. Früh getroffene Designentscheidungen werden zudem begründet.

Kapitel 4 – Smart Contract Umsetzungen eines Tabellenänderungsprotokolls: Dieses Kapitel befasst sich mit der Integration von Änderungsprotokollen in die Ethereum-Blockchain. Dabei werden mehrere potenzielle Umsetzungsvarianten und weitere Optimierungsvorschläge für die erfolgversprechendste Variante vorgestellt.

Kapitel 5 – Vergleich von existierenden Tabellenanwendungen: Mehrere existierende Tabellenprogramme werden auf ihre Erweiterbarkeit sowie die Behandlung und Auslösung von Änderungsereignissen untersucht. Die Ergebnisse des hier beschriebenen Vergleichs begründen die Entscheidung eine dieser Anwendungen für die Umsetzung zu verwenden.

Kapitel 6 – Implementierung: Fokus dieses Kapitels ist die Beschreibung von der Umsetzung der Behandlung von Tabellenänderungsereignissen, den Anstoß von Transaktionen und der Visualisierung der gespeicherten Änderungen.

Kapitel 7 – Kostenabschätzung: Dieses Kapitel widmet sich der Kostenuntersuchung von der entwickelten Anwendung. Die festgestellten Werte werden hier iteriert.

Kapitel 8 – Diskussion Die zu erwarteten Kosten und deren Implikationen werden in diesem Kapitel diskutiert. Auch auf vorhandene Einschränkungen von sowohl dem entwickelten System als auch der durchgeführten Kostenabschätzungen wird eingegangen. Anhand dieser neuen Erkenntnisse wird eine Empfehlung für kommende Arbeiten erläutert.

Kapitel 9 – Zusammenfassung und Ausblick Dieses letzte Kapitel dient einer kurzen Zusammenfassung aller in dieser Arbeit berührten Themengebiete. Abschließend werden weitere mögliche Schritte, die eine Verbesserung des entwickelten Systems oder gegebenenfalls unterschiedliche Forschungsresultate ermöglichen könnten, vorgeschlagen.

2 Related Work

Dieses Kapitel dient der Aufführung von akademischen Arbeiten, die im gegebenen Kontext von Relevanz sind. Dafür werden zuerst existierende Probleme und vorhandene Lösungsansätze präsentiert. Darauf folgt eine Einführung in die Blockchain im Generellen und Ethereum im Speziellen. Abschließend werden die gewonnenen Erkenntnisse zusammengefasst und in Verbindung mit dem Thema dieser Arbeit gesetzt.

2.1 Spreadsheets

Trotz keiner fundamentalen Änderungen innerhalb der letzten drei Dekaden haben sich Tabellen zu dem am weitesten verbreiteten Werkzeug zur Speicherung, Manipulation und Modellierung von Daten entwickelt [4]. Birch et al. [4] beschreiben unter anderem Gründe für die Verbreitung von Spreadsheetanwendungen in ihrer Voraussage der kommenden Entwicklungen. Darunter sind Faktoren wie die Offenheit, das heißt, jeder Daten- und Programmteil ist einsehbar, oder auch die einfache Erweiterbarkeit. Insbesondere die Flexibilität, also dass der Verwendungszweck bzw. Einsatzbereich von Tabellen vor einer Fokussierung auf diesen nur marginal eingeschränkt ist, ist hervorzuheben. Allerdings merken die Autoren an, dass mit wachsender Relevanz von Big Data, aber auch künstlicher Intelligenz, insbesondere Ansätze im Bereich des Machine Learning (ML), der vorhandene Zellenbereich nicht mehr ausreichen kann. Mangelnde Kernkompetenz der allgemeinen Nutzerbasis schränkt die Möglichkeiten korrekter Verarbeitung weiter ein. Auch ist es in der akademischen Literatur [25–27, 29] bekannt, dass die vorhandene Fehlerrate in Tabellen sehr hoch ist. So wird davon ausgegangen, dass ungefähr die Hälfte aller Tabellendokumente Fehler enthalten, welche mit einem wesentlichen Korrekturaufwand verbunden sind [29]. Dies beschränkt sich nicht nur auf die enthaltenen Daten, sondern auch auf höhere Features wie benutzerdefinierte Formeln oder Makros. Enthaltene Fehler können unter anderem auf menschliches Versagen zurückgeführt werden, jedoch haben die zuvor angesprochenen Skandale, welche auf mutwillige Manipulation zurückzuführen sind, zu einem höheren Bewusstsein der nötigen Kontrolle von Spreadsheets geführt. Daraus sind Regulierungen wie z. B. der Sarbanes-Oxley Act (SOX) hervorgegangen, die Fehlerriskoreduzierungsmethoden vorschreiben. Werden Verstöße gegen den vorgeschriebenen Auditierungsprozess festgestellt, sind Führungskräfte haftbar. Trotz dessen berichten Leon et al. [18], dass in vielen Firmen die verantwortlichen Prozesse Mangel aufweisen.

Jedes Spreadsheet sollte eine fest definierte Aufgabe mit klaren Verantwortlichkeitszuweisungen als auch Berechtigungen haben [19, 21]. Zusätzlich sollten neben der Funktionalitätseinschränkung je nach Rolle auch bestimmte mit hohem Risiko assoziierte Aktionen wie beispielsweise den Austausch einer Formel mit einem konstanten Wert verhindert werden [14]. Separieren von Eingabedaten, berechnender Logik und Resultaten sollte auch in Erwägung gezogen werden. Auditoren sind gefordert insbesondere Tabellen, welche Teil der finanziellen Berichterstattung sind, zu überprüfen um deren Genauigkeit und Vollständigkeit oder zumindest die durchgeführte Risikominimierung

zu garantieren. Editor und Auditor sollten zwar unterschiedliche Personen sein, jedoch sollte der Auditor über einen ausreichenden Grundkenntnisstand im Umfeld des zu überprüfenden Dokuments verfügen. Dabei ist die Änderungsanalyse nur ein Unterprozess des gesamten Audit-Prozesses. Jedoch kann eine gute Integration in effizienteren Audit-Prozessen resultieren [23]. Existiert die Möglichkeit einer Versionierung, so müssen nur Änderungen seit der letzten bestandenen Audit Analyse überprüft werden. Ebenfalls Ferreira und Visser [14] schlagen ein Versionmanagement vor, womit jede Tabellenversion gespeichert werden kann. Regelmäßige Back-Ups, in Relation zur Änderungsgeschwindigkeit eines Dokumentes, genauso wie definierte Archivierungsmethoden erachten sie als sinnvoll. Die Archivierung ist dabei ein häufig übersehener Risikofaktor [16]. Darüber hinaus fordern sie neben der Aufzeichnung von Änderungen ebenso den Zeitpunkt, Editor und Änderungsgrund festzuhalten.

Minter und Correia [21] teilen die benötigten Audit Trail-Schritte in Entscheidungsbegründung für eine Tabelle, Entwurf und Testen der enthaltenen Komponenten, initiale Datenpopulierung, in der Lebenszeit folgende Änderungen und die fortlaufende Versionskontrolle auf. Als zu jeder Tabelle zugehörige Dokumentation fordern sie den Zweck und wie dieser erreicht werden soll, die zugrundeliegenden Annahme und etwaige nachträgliche Änderungen, alle beteiligten Personen und ihre Rolle, verwandte Dokumente und Datenquellen, der zugehörige Testprozess, sowie Gründe für alle durchgeführten Änderungen zu erfassen. Genauso die Notwendigkeit von einer sicheren Speichermethodik als auch die Unabhängigkeit der Tabellen von sonstigen Veränderungen der vorhandenen IT-Landschaft wird von ihnen betont. Zugleich warnen die Autoren vor physischer Manipulation durch z. B. Kollegen.

Adler und Nash [2] entwickelten *TellTable*, welche die Tabellenanwendung in einen Web-Service extrahiert. Diese Auslagerung aller Daten und Aktionen auf einen Server begründen die Autoren mit der nicht garantierbaren Feststellung von Änderungen, falls eine lokale Zugriffsmöglichkeit besteht. Veränderungen können nur über eine Web-UI vorgenommen werden und simultane Nutzer sind auf einen pro Datei beschränkt. Das erstellte Änderungsprotokoll besteht immer neben der zugehörigen Tabellenanwendung und kann auf risikoreiche Operationen analysiert werden.

Ein weiteres von Nash et al. [23] entwickeltes Stand-Alone Audit Werkzeug, kann Änderungen einer *LibreOffice Calc*-Tabelle aufgezeichnet und diese filtern. Die Aufzeichnung ist nicht mit der eigentlichen Tabellenanwendung verbunden, sondern ein externes Programm überwacht Dateien des File-Systems. Dokumentiert werden dabei der Zeitpunkt, Editor, vorherige Wert und neue Wert. Wie aus Abbildung 2.1 erkennbar ist, werden diese zeilenbasiert wiedergegeben. Dies erlaubt die vollständige Rekonstruktion einer Tabelle. Die Autoren merken an, dass durch den Zugriff von Nutzern in deren eigener Umgebung eine Kontrolle nicht zwingend gegeben ist und böswillige Manipulationen nicht auszuschließen sind. Sie schlagen eine Zentralisierung der Tabellendokumente mitsamt Zugriffskontrollsystem vor. Zusätzlich fordern sie eine Verschlüsselung der Daten während der Komponentenkommunikation, um Verfälschungen der Daten oder die Herausgabe von sensitiven Informationen zu verhindern.

Die Nutzung von Makros stellt eine schwierige Kosten-Nutzen-Abwägung dar. Im passenden Kontext eingesetzte und durchdacht entwickelte Makros können den Arbeitsaufwand deutlich verringern, jedoch sind sie eine häufige Fehlerquelle. Zumal keine allgemeingültige Lösung für die Entwicklung und Verwaltung von Tabellen existiert [10, 17], sollten diese und andere Limitierungen daher mit allen betroffenen Parteien abgesprochen werden, wie Ferreira und Visser [14] zu bedenken geben. Die Autoren heben die Befürchtung der Unternehmensführung vor dem Verlust der eigentlichen

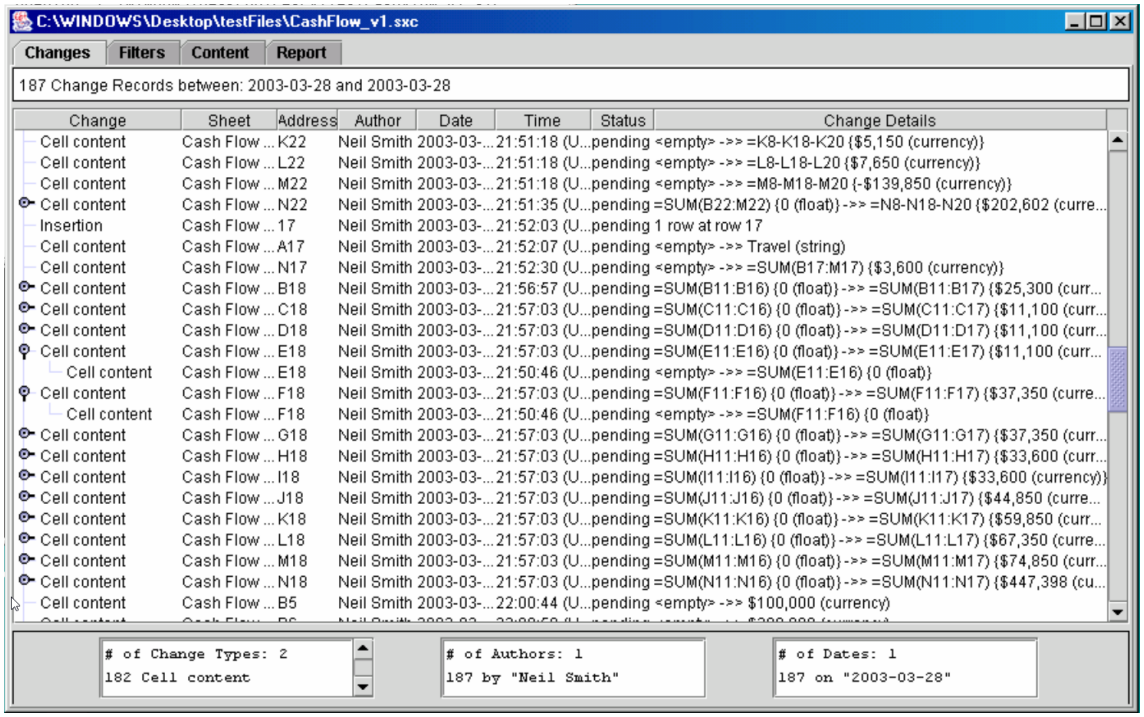


Abbildung 2.1: Benutzeroberfläche des Änderungsprotokolls von Nash et al. [23]

Änderungen werden nach Zelle gruppiert und zeilenbasiert wiedergegeben. Diese sind dabei um Metadaten wie Änderungszeitpunkt und Editor sowie eine Klassifikation ergänzt worden. Filter zur einfacheren Überprüfung existieren in einem anderen Fenster.

Spreadsheet-Effizienz durch Einführung von vielen Kontrollmechanismen hervor. Auch betonen sie die Bedeutung von einer systematischen Vorgehensweise, welche zu großen Teilen proaktiv Risiken minimieren kann.

Birch et al. [4] merken auch an, dass durch die Einführung von Software-As-A-Service, gerade bei der Ursprungsverfolgung von Änderungen Herausforderungen entstehen. Für Versionierung und garantierte Integrität sind neue Prozesse nötig, die den gleichzeitigen Zugriff gegebenenfalls beschränken. Diese Auslagerung in die Cloud reduziert die Neu- oder Weiterentwicklung und das Management der firmeneigenen IT-Umgebung. Die Autoren stellen in Aussicht, dass neue spezialisierte Tools für möglicherweise erneuertes Fachpersonal entwickelt werden wird. Allerdings könnte diese Entscheidung aktuell, gegeben der bereits vorhandenen Familiarität mit Tabellenprogrammen, noch nicht kosteneffektiv und risikobehaftet sein [14, 19]. Sie heben unter anderem einen wahrscheinlichen Wandel zu verbesserter Unterstützung für Inspektionswerkzeuge für alle bestehenden Tabellenprozesse. Genauer umschließt dies das Design, die bereitgestellte Nutzerunterstützung um das Verständnis von Ziel und Funktionsweise zu erhöhen sowie die Kontrolle. Eine weitreichende Automatisierung der Überprüfung der verantwortlichen Datenpipeline ist ein praktischer Ansatz. Insbesondere die mit der Cloud einhergehende kollaborative Mehrbenutzerfunktionsweise erfordert die Unterstützung von robusten Versionskontrollsystemen, wie in der Softwareentwicklung bereits lange erprobt. Allerdings äußern Adler und Nash [2] Bedenken, dass die Identifizierung, das Zusammenführen wie auch die Konfliktbehandlung noch nicht ausreichend erforscht ist.

2.2 Blockchain

Kryptowährungen und Blockchain zählen zu den aktuellen Modewörtern und technologischen Trends, welche in vielen unterschiedlichen Themengebieten neu erprobt werden und das Potenzial haben, diese zu transformieren [35]. Diese Entwicklung kann dem Erfolg von Bitcoin zugerechnet werden, jedoch existieren noch keine exakt definierten Begrifflichkeiten und genauen Abgrenzungen [20]. Prinzipiell kann die Blockchain als fortlaufende Liste verstanden werden, in der Einträge, genannt Blöcke, in einer Abhängigkeitsbeziehung zu dem vorhergegangenen Eintrag stehen. Blöcke können dementsprechend nur hinten angefügt werden. Alle Blöcke sind größenbeschränkt und haben damit ein oberes Limit an enthaltenen Daten. Zheng et al. [35] geben den Trade-Off von einer zu freien Blockgrößenerhöhung zu bedenken. Diese benötigten zwingend mehr Speicherplatz und sind langsamer in der Propagierung. Vor einer Reduktion der Nutzer, die die Möglichkeit haben und auch gewillt sind diese Anforderung zu erfüllen, wird gewarnt. Aus einer geringeren Nutzerzahl folgt eine zunehmende Zentralisierung der Blockchain, woraus ein erhöhtes Manipulationsrisiko resultiert. Diese kann sich auch durch die eigene Popularität ergeben, da die Blockchain nicht linear skaliert. Die zu verarbeitenden Nachrichten wachsen schneller als die Anzahl der Teilnehmer, womit die benötigten Ressourcen wiederum ansteigen [3, 5]. Gegebenenfalls können auch nicht mehr alle Nachrichten zeitnah bearbeitet werden [35].

Grundeigenschaften der Blockchain sind ein dezentralisierter Aufbau, Persistenz, Pseudoanonymität und Nachvollziehbarkeit (Auditability) [35]. Dezentralisierung wird durch viele konkurrierende Akteure, welche einen Konsens suchen, erreicht. Es existiert damit kein Single Point of Failure (einzelner Ausfallpunkt) (SPOF). Jeder Akteur führt eine eigene Instanz aus und ist damit ein Knoten des Netzwerkes. Akteure werden durch zugehörige Adressen identifiziert. Zwischen Adresse und realer Identität besteht kein direkter Zusammenhang, jedoch ist es nicht auszuschließen, dass dieser intrinsisch, anhand vergangener Transaktionen, hergestellt werden kann. Akteure, die aktiv Transaktion verarbeiten, um neue Blöcke zu generieren, werden Miner genannt. Anreiz dafür ist die Kompensation mit Tokens, welche meistens als Zahlungsmittel fungieren. Tokens einer Blockchain-Implementierungen bzw. Instanz haben keinen Wert auf einer anderen. Der angesprochene Konsens wird über einen Algorithmus ermittelt. Die genaue Funktionsweise ist dabei nicht über verschiedene Blockchains einheitlich. Er dient einzig der Konsensfindung und ist unabhängig von Transaktionen [3]. Proof-of-Work (PoW) ist ein populärer Ansatz, welcher schwer zu berechnen, allerdings leicht zu überprüfen ist [3, 35]. Proof-of-Stake (PoS) ist nicht von der Lösung einer schweren Aufgabe abhängig und damit energiegünstiger als PoW. Dieser Algorithmus basiert auf der Annahme, dass die Wahrscheinlichkeit, dass Akteure, die viele Tokens besitzen, das Netzwerk angreifen, geringer ist [35].

Eine Kategorisierung von Blockchain ist in drei grobe Gruppen möglich [7]. Entscheidendes Kriterium ist die Einschränkung der aktiven Teilnehmer, der es erlaubt ist neue Blöcke anzufügen und über deren Validität zu entscheiden.

Private Blockchain: Sind alle beteiligten Akteure aus der gleichen Organisation, wird dies als private Blockchain bezeichnet. Die Dezentralisierung ist hier nicht mehr vorhanden.

Konsortium Blockchain: umfasst Netzwerke, die aus einem Zusammenschluss von Knoten mehrerer unterschiedlicher Organisationen entstanden sind. Dabei haben nur diese Knoten das Recht neue Blöcke anzufügen. Diese Kategorie erfüllt die Dezentralisierungseigenschaft teilweise.

Öffentliche Blockchain Hat jede Person die Möglichkeit sich aktiv zu beteiligen, gilt dies als öffentliche Blockchain.

Alle vorherigen und, falls nicht explizit anders benannt, nachfolgenden Angaben basieren auf diesem Umsetzungsmodel. Unabhängig der Einordnung, kann das Netzwerk öffentlich einsehbar und lesbar sein, jedoch ist dies nur in der öffentlichen Blockchain zwingend nötig. Eine steigende Einschränkung der Knotenmenge resultiert im Verlust von Blockchain-Eigenschaften. Private und Konsortium Blockchain werden manchmal auch unter permissioned Blockchain zusammengefasst.

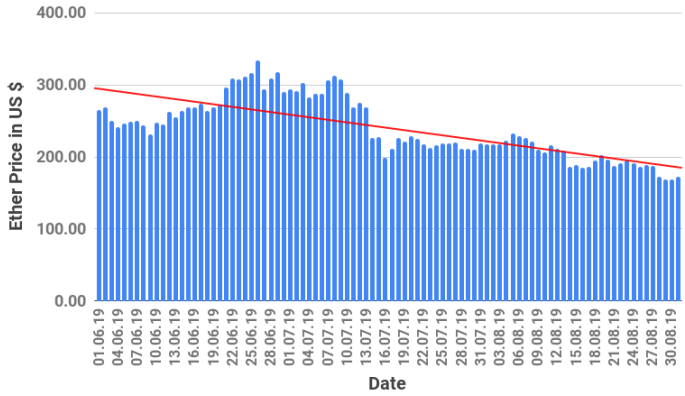
Kommunizieren Akteure untereinander, um beispielsweise Tokens zu übertragen, wird dieser Vorgang Transaktion genannt. Ein privater Schlüssel wird verwendet um Daten zu signieren und dann an das Netzwerk zu übertragen. Diese können mit dem öffentlichen Schlüssel und den ursprünglichen Daten auf Manipulation überprüft werden [35]. Eine nachträgliche Änderung von Transaktionen ist demnach nicht möglich [28]. Gerade in Geschäftsbereichen, welche Zuverlässigkeit und die Ehrlichkeit aller teilnehmenden Partei erfordern, ist eine Integration der Blockchain für beide Parteien attraktiv [35]. Ammous [3] beschreiben die Entscheidung für eine Blockchain-Technologie damit, dass diese nicht gewählt wird, weil sie viele günstige Transaktionen erlaubt, sondern obwohl sie dies nicht unterstützt. Das nicht benötigte Vertrauen in andere Parteien ist, insbesondere für monetäre Transaktionen, der Hauptanziehungsfaktor. Diese, im Zusammenhang mit der Blockchain, häufig genannte Unveränderbarkeit, ist allerdings nur praktisch gegeben. Es kann lediglich ein Manipulationsresistenz (tamper resistance) festgestellt werden. Sobald ein Angreifer 51% verarbeitenden Ressourcen bereitstellt, kann er alleine den Konsens bestimmen [3, 35]. Diese Gefahr besteht hauptsächlich in kleineren Blockchains. Allerdings kann auch die Mehrheit der Community zusammen mit den Begründern zu einem Rückgängigmachen entschließen. Ein Beispiel dafür ist die damalige umstrittene Gabelung (Fork) von Ethereum in Ethereum und Ethereum Classic im Juli 2016 [6]. Nachdem durch einen Exploit ungefähr US$50 000 000 aus einem Risikokaptialfonds (decentralized autonomous organisation (DAO)[1]) entwendet wurden, entschied sich die Mehrheit der Ethereum-Community zu einem festgelegten Zeitpunkt auf einen vorherigen Block zurückzurollen. Ethereum Classic hat diesen Rollback nicht vollzogen, denn der mit diesem Exploit verbundene Code war für alle Investoren vor ihrer Beteiligung einsehbar und sie hatten den gegebenen Bedingungen, wenn auch unwissentlich, zugestimmt [3].

2.2.1 Ethereum

Heutzutage existieren viele verschiedene Blockchain-Implementierungen mit ihren jeweiligen Eigenheiten. Eine genauere Erläuterung der grundlegenden Prinzipien erfolgt am Beispiel von Ethereum. Nach dem initialen Erfolg von Bitcoin entstand Ethereum mit dem Ziel einer Turing-vollständigen Blockchain [5]. Für die Ausführung auf der Blockchain entwickelte Programme werden *Smart Contract* genannt. Ethereum ist momentan die größte Blockchain, die diese unterstützt[2]

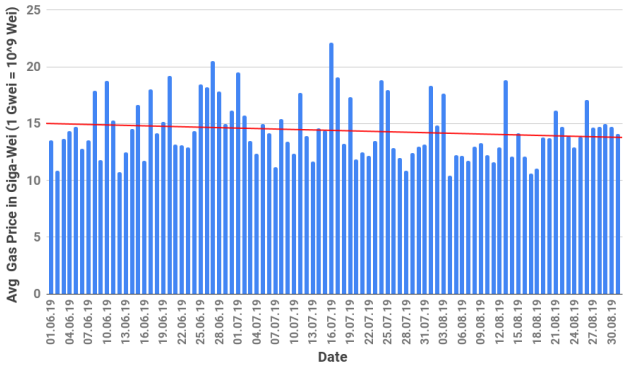
Ethereum-Adressen, auch Accounts genannt, sind ein 40 Zeichen langer Hexadezimal-String, der eine eindeutige Identifizierung erlaubt. Accounts können in zwei prinzipielle Funktionszwecke aufgeteilt werden. Ersterer ist einem Akteur zu repräsentieren und speichert Ethereum-Tokens, auch Ether (ETH) bezeichnet. Diese Tokens werden als Währung genutzt. Dabei entspricht ein ETH 10^{18}

[1] https://www.investopedia.com/tech/what-dao/
[2] https://coinmarketcap.com/all/views/all/



(a) Preisverlauf des Ethereum Token

Durchschnitt des täglichen Wechselkurses zwischen einem Ethereum Token ETH und US$



(b) Preisverlauf einer Einheit *Gas*

Durchschnittlich bezahlte Ethereum Tokens ETH pro Einheit *Gas* an einem Tag | 1 ETH = 10^{18} Wei & 1 Giga-Wei (Gwei) = 10^9 Wei

Abbildung 2.2: Preisentwicklung der Ethereum-Einheiten von Juni 2019 bis September 2019

Wei der kleinstmöglichen Darstellung dieser Einheit. Abbildung 2.2a zeigt die Entwicklung des Wechselkurses zwischen einem ETH und US$ zwischen Juni 2019 und September 2019 (M = 240,29; SD = 42,37). Die Identifikation von Smart Contracts ist der zweite Verwendungszweck. In diesem Fall sind zusätzlich die verbundenen Ausführungsanweisungen und gegebenenfalls vorhandene Variablen Teil der gespeicherten Daten.

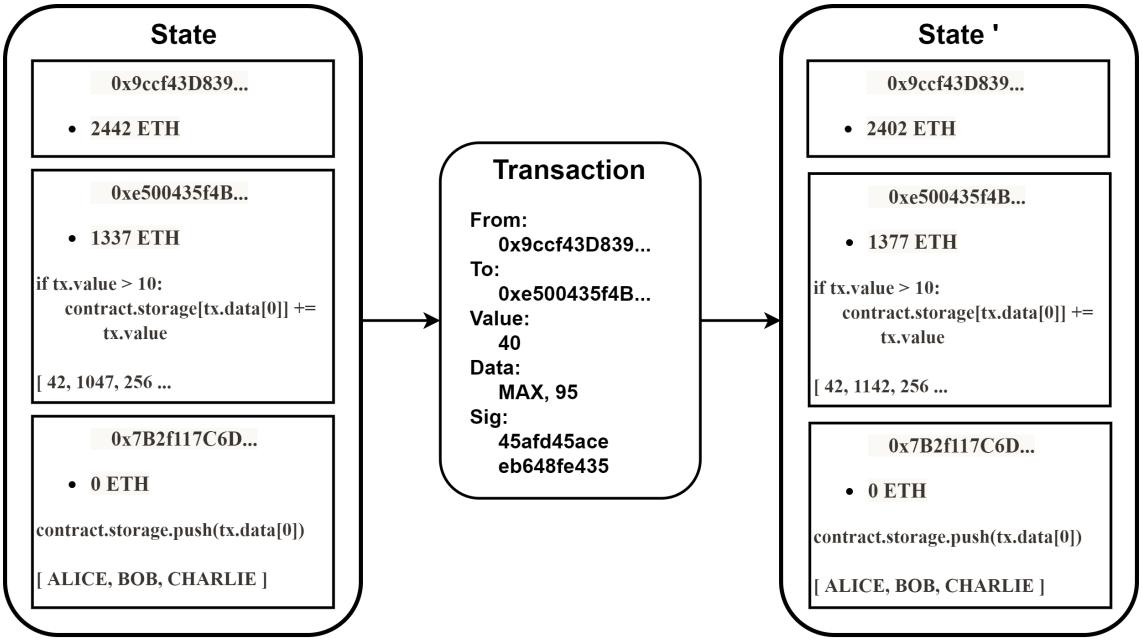
Smart Contracts

Selbst definierte Operationen, die innerhalb eines Ethereum-Knotens ausgeführt werden, sind Teil eines Smart Contracts, welcher die Menge der mit einer Adresse verbundenen Operationen umschließt. Da die Exekution auf Ethereum-Knoten passiert, gelten alle Blockchain-Eigenschaften für diese Anwendungen. Mehrere Programmiersprachen stehen für die Entwicklung zur Verfügung.

Solidity[3] ist dabei momentan die wohl populärste, am weitesten entwickelte, sowie am besten dokumentierte Programmiersprache. Syntaxisch wurde Solidity von *C++*, *Python* und *JavaScript* beeinflusst. Ein Beispiel, eines Smart Contracts in Solidity, ist in Listing 2.1 aufgeführt. In dieser primitiven Anwendung wird ein Integer-Wert zwischen 0 und 2^{256} mit einer Adresse verknüpft. Der aktuellste Wert von *storedData* ist immer Teil der Blockchain. Zusätzlich existieren Methoden, die eine Manipulation und die Abfrage des aktuellen Wertes zulassen.

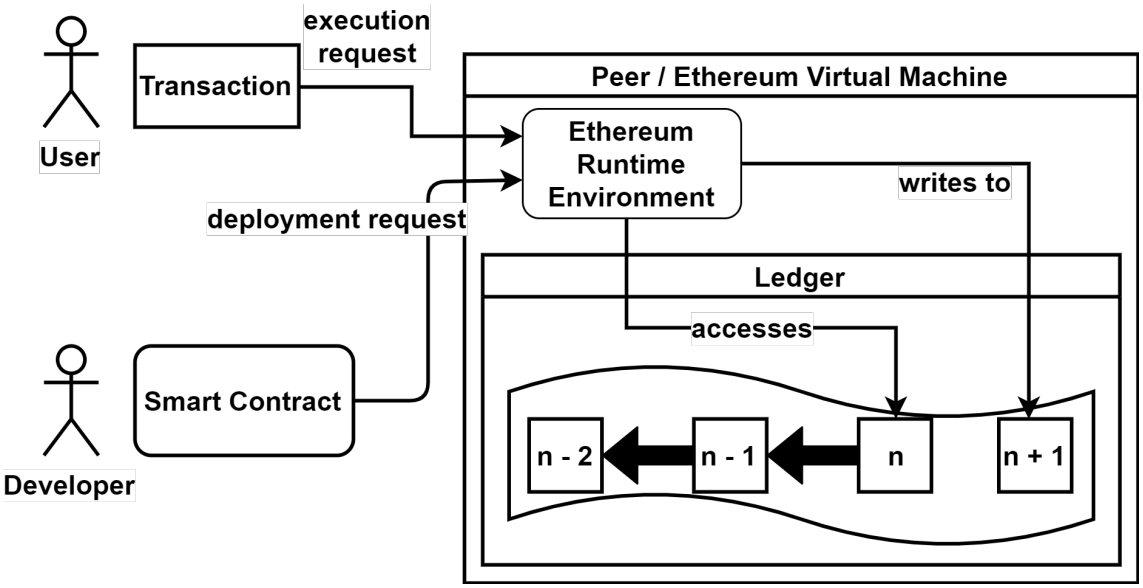
Um eine Ausführung auf einem Ethereum-Knoten zu ermöglichen, müssen Smart Contracts in für die Ethereum Virtual Machine (EVM) verständliche Anweisungen umgewandelt werden. Diese Anweisungen werden fortan als OPCodes bezeichnet und sind beispielhaft für die *get* und *set* Methoden aus Listing 2.1 in Listing 2.2 dargestellt. Jede Ausführung eines OPCodes erzeugt Kosten, welche in *Gas* angegeben werden. *Gas*-Kosten für den gleichen OPCode sind bis zu einer Protokollanpassung der Blockchain immer identisch. Eine Auswahl an Operationsgruppen und deren Kosten ist in Tabelle 2.1 aufgelistet. Dabei ist auffällig, dass einige Operationen signifikant teuer sind. Diese sind in der Regel eng mit Ethereum-Konzepten verwandt. Auch existiert die Möglichkeit negative Kosten, also eine Gutschrift, zu verursachen. Der Datenaustausch zwischen zwei Accounts, also auch Aufruf von Smart Contract-Methoden, ist ein Beispiel einer Transaktion. Als Anreiz, dass Miner Transaktionen verarbeiten, muss ein Betrag angegeben werden, der für jede verbrauchte Einheit *Gas* bezahlt wird. Ein Mindestbetrag, unter welchem Miner Transaktionen nicht bearbeiten, existiert durch das eigennützige Vorgehen der Miner, welche Transaktionen mit potenziell höherem Ertrag priorisieren. Dementsprechend ist der zu zahlende Betrag marktwirtschaftlich geregelt. Sind mehrere Miner nahezu gleichzeitig erfolgreich, entstehen *Onkel*- oder auch *Ommer*-Blöcke. Bis zur siebten Generation werden Miner für die Weiterführung dieser entschädigt [5]. Die Einteilung in *Onkel* oder Hauptkette erfolgt durch die Fortführung der aktuell längsten Kette. Die zuvor genannte Größenbeschränkung eines Blockes ist über das maximal enthaltene *Gas* geregelt. Momentan ist das obere Limit von Ethereum 6 721 975. Reicht das Guthaben eines Accounts nicht für die Fortführung einer Transaktion aus oder wird das angegebene *Gas*-Limit erreicht, wird diese abgebrochen. In diesem Fall werden alle gemachten Änderungen verworfen und der anstoßende Account trotzdem voll belastet. Zwischen Juni und September 2019 [13] lag dieser im Durchschnitt bei ungefähr 14 400 489 878 (Standardabweichung (SD): 2 138 269 272) Wei, was bei dem zuvor genannten ETH Preis US$0,000 003 460 29 entspricht. Der Verlauf des verbundenen *Gas*-Preises in diesem Zeitraum ist, wie in Abbildung 2.2b zu sehen, leicht rückläufig.

Chen et al. [8, 9] untersuchten diese *Gas*-Kosten. Sie beschreiben mehrere kostspielige Anti-Pattern und zeigen, dass ein signifikanter aller eingesetzten (deployed) Smart Contracts diese enthalten. Es gelang ihnen eine Korrelation zwischen der Größe eines Smart Contracts und der Enthaltungswahrscheinlichkeit eines Anti-Patterns nachzuweisen. Aufgrund der Unveränderbarkeit ist eine nachträgliche Verbesserung nicht möglich. Dementsprechend ist Überprüfung und Korrektur während der Entwicklung wichtig.



(a) Übergang zwischen zwei Ethereum-Zuständen durch eine Transaktion [5]

Manipulation des World States durch eine Transaktion. Diese übertragt 40 ETH und addiert einen mitgelieferten Wert mit dem Inhalt einer Map.



(b) Interaktion mit der Ethereum-Blockchain durch die EVM

Akteure interaktieren mit der EVM. Diese wiederum führt die angefragten Operationen in ihrer Ethereum Runtime Environment (Laufzeitumgebung) (RE) aus. Dabei kann sie Daten von bisherigen Blöcken lesen oder neue Daten in einen neuen Block schreiben. Die ursprüngliche Transaktion, welche die schreibende Operation auslöste, wird an andere Peers propagiert.

Abbildung 2.3: Ethereum-Übergänge und deren Anstoß

Listing 2.1 Einführungsbeispiel von Solidity[4]

```
pragma solidity >=0.4.0 <0.7.0;

contract SimpleStorage {
  uint storedData;

  function set(uint x) public {
    storedData = x;
  }

  function get() public view returns (uint) {
    return storedData;
  }
}
```

Listing 2.2 Teil der aus Listing 2.1 erstellten OPCodes

```
tag 7               function set(uint x) public {...
  JUMPDEST          function set(uint x) public {...          1 Gas
  DUP1              x                                         3 Gas
  PUSH 0            storedData                                3 Gas
  DUP2              storedData = x                            3 Gas
  SWAP1             storedData = x                            3 Gas
  SSTORE            storedData = x                            20 000 or 5 000 Gas
  POP               storedData = x                            2 Gas
  POP               function set(uint x) public {...          2 Gas
  JUMP [out]        function set(uint x) public {...          8 Gas
tag 9               function get() public view returns...
  JUMPDEST          function get() public view returns...     1 Gas
  PUSH 0            uint                                      3 Gas
  DUP1              storedData                                3 Gas
  SLOAD             storedData                                200 Gas
  SWAP1             return storedData                         3 Gas
  POP               return storedData                         2 Gas
  SWAP1             function get() public view returns\{...   3 Gas
```

Ethereum Virtual Machine (EVM)

Aufgrund der Dezentralisierung existieren viele gleichgestellte Akteure (Peers). Mehrere mögliche Clients, wie beispielsweise Geth[5], können als Knotenpunkt fungieren. Um gleiche Funktionalität zu garantieren, ist das zugrundeliegende Ethereum-Model formal definiert. Dieses Model ist als EVM bekannt und beschreibt eine Stack-Architektur einer State Transition Machine [33]. Die Verarbeitung

[3] https://solidity.readthedocs.io/en/latest/index.html
[4] https://solidity.readthedocs.io/en/v0.5.12/introduction-to-smart-contracts.html#storage-example
[5] https://geth.ethereum.org/

Operationsgruppe	*Gas*-Kosten in Wei	Beschreibung
$G_{transaction}$	21000	Transaktionsbeginngebühren
$G_{txdatazero}$	4	Pro enthaltenem Null-Byte einer Transaktion
$G_{txdatanonzero}$	68	Pro enthaltenem nicht Null-Byte einer Transaktion
G_{call}	700	Interaktion mit einem Ethereum Account
G_{sload}	200	Lesen aus dem World State Speicher
G_{sset}	20000	Update eines Nullwerts im World State zu nicht-null
G_{sreset}	5000	Änderung eines Wertes im World State der davor nicht null war
R_{sclear}	15000	Rückerstattung pro 32 Bytes Wort, das im World State nullgesetzt wird und davor nicht null war
$G_{newaccount}$	25000	Erstellen eins Accounts
G_{log}	375	Erstellen eines Logs
G_{logdata}	8	Pro enthaltenem nicht indexierten Byte eines Logs
G_{logtopic}	375	Pro enthaltenem Topic eines Logs
G_{zero}	0	umfasst die OPCodes {STOP, RETURN, REVERT}
$G_{jumpdest}$	1	umfasst den OPCode {JUMPDEST}
G_{base}	2	umfasst die OPCodes {ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, RETURNDATASIZE, POP, PC, MSIZE, GAS}
$G_{verylow}$	3	umfasst die OPCodes {ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}
G_{low}	5	umfasst die OPCodes {MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}
G_{mid}	8	umfasst die OPCodes {ADDMOD, MULMOD, JUMP}
G_{high}	10	umfasst den OPCode {JUMPI}

Tabelle 2.1: *Gas*-Kosten ausgewählter Operationen, vollständige Tabelle in [33]

von Transaktionen führt, wie in Abbildung 2.3a skizziert, zu neuen Zuständen. Transaktionen werden von Accounts angestoßen und enthalten unter anderem einen Empfänger, die Anzahl der zu sendenden Tokens, den zu zahlenden Preis pro *Gas*-Einheit, die maximale *Gas*-Menge und eine Signatur des Senders. Einzig bei der Einstellung (deployment) eines Smart Contracts wird keine Adresse angegeben. Die allgemeine Vorgehensweise ist in Abbildung 2.3b dargestellt. Ein Account kann mit einer Transaktion an die EVM herantreten. Diese ist bei der Verarbeitung von der Außenwelt, mit Ausnahme des Blockchain-Ledgers, isoliert. Dabei kann sie auf die im letzten Blockchain-Block referenzierten Daten zugreifen und diese für die Erstellung des nächsten Blocks verändert. Ein schemenhafter Aufbau der Blockchain ist in Abbildung 2.4 skizziert. Jeder Block besteht aus einem Header der Informationen, die für die Verifizierbarkeit benötigt werden, und

drei modifizierte Merkle-Patricia-Bäume enthält. Diese Tries ermöglichen die authentifizierbare Speicherung von Daten. Der erste Trie wird *World State* genannt und hat Ähnlichkeit mit einer Datenbank [9]. Er enthält alle mit Accounts verbundenen Informationen. Der Transaktions-Trie ist der zweite Trie. Hier sind alle in diesem Block bearbeiteten Transaktionen enthalten. Informationen zu den bearbeiteten Transaktionen, wie entstandene Logs oder verbrauchtes *Gas*, werden im dritten Trie gehalten. Dieser wird als Transaktionsquittungs-Trie bezeichnet. Eine vollständige Definition der enthaltenen Felder ist in [33] aufgeführt.

Eine der zuvor angemerkten mehrdeutigen Begrifflichkeiten existiert für *On-Chain* und *Off-Chain*. So können diese den Speicherort von Daten [1], als auch den Ausführungsort beschreiben [30]. Vollständig in einem der Tries enthalte Daten gelten als *On-Chain*, ist nur der Hash der Daten, gegebenenfalls als Verweis auf den Block einer permissioned Blockchain, vorhanden, gilt dies als *Off-Chain* Data Storage Solution. Wird ein System in *On-Chain* und *Off-Chain* geteilt, dann bezieht sich der *On-Chain*-Teil auf die Funktionen die in einer EVM ausgeführt werden. Als *Off-Chain* werden die Programmteile referenziert, für die dies nicht gültig ist.

2.2.2 Anwendungsbereiche

Aktuell wird die Blockchain größtenteils im finanziellen Sektor eingesetzt, jedoch dringt die Blockchain-Technologie in immer weitere Geschäftsfelder ein [35]. So existiert eine Blockchain-basierte Anti-Malware-Umgebung [24], eine Verfolgung und Bewertung von erarbeitetem Ansehen (reputation) [32] und die Verbindung mit Internet of Things (Internet der Dinge) (IoT) [34].

Applikationen, die sich die Ethereum-Blockchain zunutze machen, können noch in drei grobe Kategorien eingeteilt werden [5]. Finanziell motivierte Anwendungen stellen die erste Gruppe dar und bieten hauptsächlich verwaltende Funktionen an. Nicht finanziell gestaltete Applikationen, wie Online-Voting, sind die zweite Gruppe. Die dritte Kategorie wird aus der Mischung der ersten beiden erstellt. Das heißt, dass ein signifikanter Anteil von dem finanziellen Aspekt befreit ist, aber Geld trotzdem weiterhin benutzt wird. Ein Beispiel wäre der Kauf von korrekt gelösten mathematischen Problemen (computational problems). Der erste Akteur der eine korrekte Lösung für ein ausgeschriebenes Problem präsentiert, wird für seinen Aufwand entschädigt.

Die Nutzung der Blockchain als Speicherort ist ein aktuelles Forschungsgebiet. Eine Kombination der Blockchain-Technologie mit Big Data, sowohl zur Analyse als auch zum Management, genauer der Speicherung von wichtigen Daten, erscheint möglich [35]. So kann die Lieferkette von Lebensmitteln mittels Blockchain-Transaktionen nachverfolgt werden [22]. Ein Ansatz zur Speicherung und Verifizierung von Arbeitsqualifikationen und Erfahrungen wird von Sarda et al. [31] vorgestellt. Ramachandran und Kantarcioglu [30] nutzen die Blockchain um Änderungen in dem Datenbereich von Untersuchungsergebnissen zu kontrollieren und verifizieren. Ihre vorgeschlagene Lösung erfordert „konstante Kosten und moderaten Overhead“ [30]. Allerdings werden die Daten in einem Cloud-Speicher und nicht auf der Blockchain gehalten, jedoch sind Änderungsprotokolle über Ethereum-Logs verfügbar. Buterin [5] merken an, dass ein Smart Contract mit dem Nutzer Teile ihres lokalen Speicherplatzes vermieten können, in der Größenordnung von $20 - 200$ GB ökonomisch sinnvoll seinen könnte.

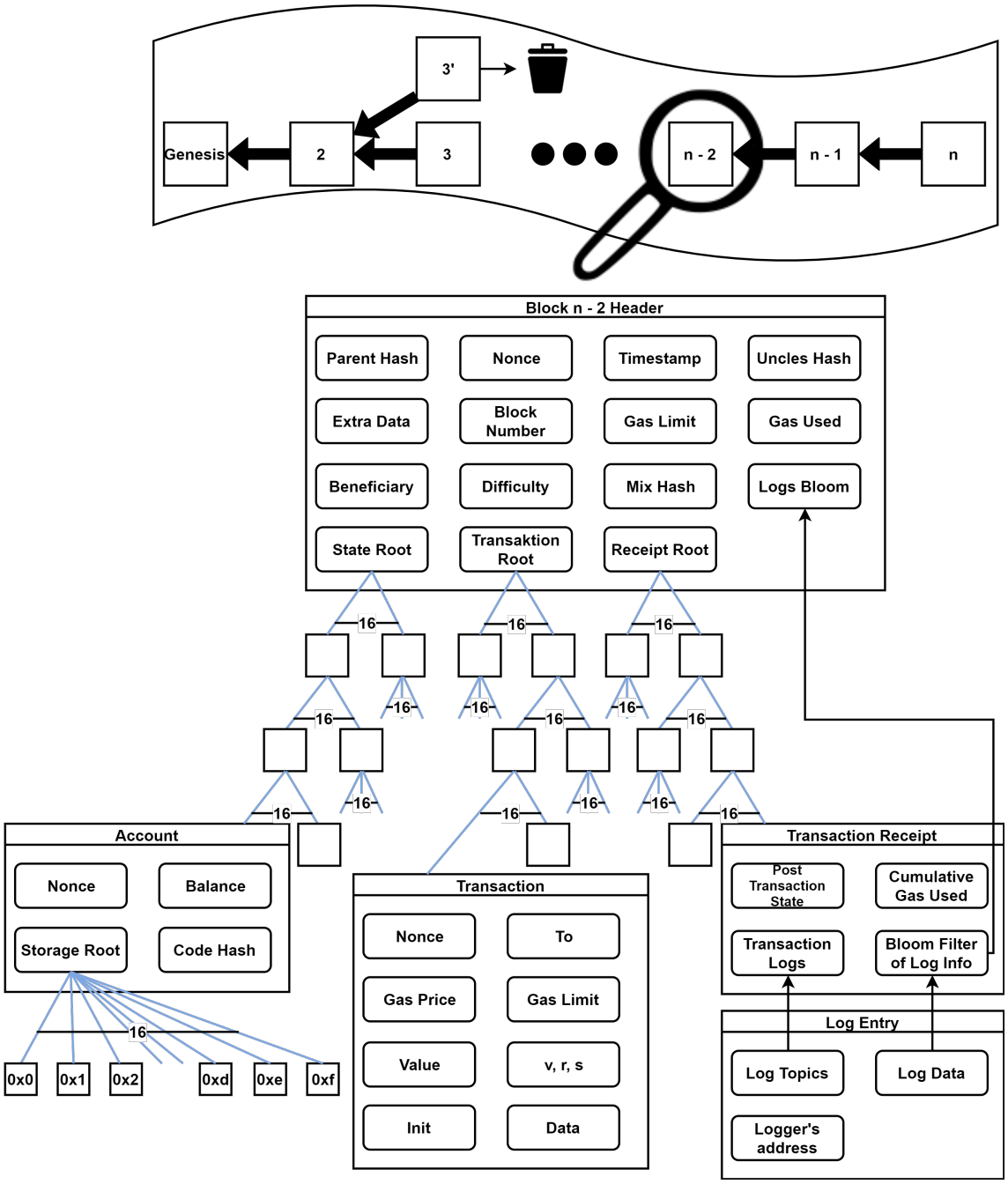


Abbildung 2.4: Vereinfachter Aufbau der Blockchain

Der erste Block wird Genesis-Block genannt und wird für die Synchronisierung der Blockchain benötigt. Danach wird die längste Kette fortgeführt. Ein Block speichert Metadaten in Headern. Zusätzlich enthält er die Wurzeln (Root) der drei enthaltenen Tries. Diese haben bis zu 16 Kinder und enthalten den World State und die bearbeiteten Transaktionen mitsamt deren Ergebnis. Zudem ist ein Bloomfilter zur effizienten Suche vorhanden.

3 Konzept

Verwandte Arbeiten zeigen, dass Auditdurchführungen einige Fehlerrisiken von Tabellen verringern können. Da die dafür nötigen Änderungsprotokolle allerdings sowohl während der Erstellung, als auch im Nachhinein geändert werden können, ist eine manipulationssichere Speicherlösung nötig. Die Blockchain bietet diese Eigenschaften, jedoch wurde noch keine Umsetzung eines Audit Trails auf der Blockchain gefunden. Eine Untersuchung der mit diesem Ansatz verbundenen Kosten erscheint sinnvoll. Das folgende Kapitel beschreibt zuerst die gesetzten Ziele gefolgt von der Begründung früh getroffener Designentscheidungen. Abschließend werden noch die Einschränkungen des gewählten Ansatzes und der getroffenen Entscheidungen diskutiert.

3.1 Entwurfsziele

Die Ziele können in die folgenden Kategorien aufgeteilt werden:

Zuverlässigkeit: Um ein vertrauenswürdiges Änderungsprotokoll zu erreichen, müssen vom Nutzer durchgeführte Änderungen automatisch erfasst werden.

Nutzerfreundlichkeit: Der vorherige Punkt dient bereits sekundär einer Erhöhung der Nutzerfreundlichkeit. Zusätzlich nötige Nutzeraktivität soll möglichst gering sein und die Programmfunktionalität in unabhängigen Dokumenten nicht eingeschränkt werden. Auch soll die Aktivierung möglichst unkompliziert und für bereits existierende Dokumente möglich sein. Eine einmalige Initialisierung eines Dokuments soll ausreichen.

Reaktives Verhalten: Sollen gemachte Änderungen gespeichert werden, erfordert dies zwingend die Persistierung auf der Ethereum-Blockchain. Dementsprechend sollte das lokale Speichern nach Möglichkeit mit der übertragenden Transaktion verbunden werden. Diese Transaktionsauslösung soll aufgrund eines reaktiven Kriteriums angestoßen werden.

Modularität: Damit ist die Aufteilung in eine *externe* Anwendung zur Datenaggregation und einer Blockchain-nativen Applikation zur Persistierung gewünscht. Für die Kommunikation zwischen diesen beiden Anwendungsteilen ein lokaler Ethereum-Knoten vorgesehen. Eine öffentliche Blockchain ist wünschenswert, um die Blockchain-Eigenschaften und damit verbundenen Vorteile sicherzustellen.

Wirtschaftlichkeit: Unabhängig von dem Gesichtspunkt der Kostenanalyse sollen die Betriebskosten des entwickelten Produkts minimal sein. Dabei wird der benötigte Speicherplatz voraussichtlich den primären Kostenfaktor der Ethereum-Transaktionen und dem infolge des ganzen Speicherprozesses darstellen. Dies bedeutet, dass sowohl eine minimale Darstellung der Änderungen als auch möglichst wenige Transaktionen gewünscht sind.

Erweiterbarkeit der Auditierfunktionalität: Die gespeicherten Änderungsprotokolle abzurufen sollte mit einem möglichst geringen Aufwand verbunden sein, damit fachlich bedingte Erweiterungen ohne großen Aufwand hinzugefügt werden können. Auch sollten keine Vorbedingungen daran geknüpft sein. Für diese Anforderung ist eine Umsetzung als eigenständige Webkomponente sinnhaft. Die Visualisierung aller Änderungen ist für eine effiziente Durchführung von Audit Trails zwingend notwendig. Weitere Überprüfen sind in ihrer Nützlichkeit nicht allgemeingültig, weshalb sich anfangs auf eine reine Visualisierungskomponente beschränkt wird.

3.2 Blockchain-Transaktionsauslösung

Für die Auswahl des Zeitpunkts, zudem die erfassten Änderungen auf die Ethereum-Blockchain geschrieben werden sollen, ist eine genauere Betrachtung der möglichen Auslöser einer Transaktion nötig. Diese können in zwei grobe Kategorien eingeteilt werden. Erstere löst für jede Änderung eine Transaktion aus. Dies bedeutet die höchstmögliche Anzahl an Transaktionen. Weitere Optimierungen der Daten sind gegebenen ihrer Minimalität nicht möglich. Zusammen mit der Bepreisung des Transaktionsanstoßes sind vergleichsweise hohe Kosten zu erwarten. Dieser Ansatz hat darüber hinaus ein erhöhtes Fehlerrisiko, durch die einhergehende hohe Anzahl an Transaktionen. Insbesondere die fehlende Garantie einer Beibehaltung der Transaktionsreihenfolge kann selbst ohne abgelehnte Transaktionen zu fehlerhaften Änderungsprotokollen führen. Neue Transaktionen könnten daher erst nach dem erfolgreich bestätigten Abschließen der vorherigen Transaktion angestoßen werden. In Verbindung mit der atomaren Speicher-Operation, sind, selbst bei einer 100 prozentigen Auswahlquote der Miner, viele lange Sperr- bzw. Wartezeiten zu erwarten. Die Alternative zu sofortigen Transaktionen ist die Sammlung von Änderungen in einer Massentransaktion. Dies zweite Variante erlaubt eine Reduktion der Datengröße, da lediglich eine Referenz auf das Dokument nötig ist, anstelle diese Referenz jeder Änderung beifügen zu müssen. Eine weitere Optimierung der zu speichernden Datengröße kann durch die Minimierung auf den letzten geänderten Wert in einer Zelle erreicht werden. Dabei kann die Transaktion via eines frei gewählten Kriteriums, beispielsweise Zeit seit letzter Transaktion, Anzahl zwischengespeicherter Änderungen oder erwartete Transaktionskosten, einer fest definierten Aktion, wie dem Schließen oder Speichern eines Dokuments, oder durch gezielte Benutzerinteraktion angestoßen werden. Eine beliebige Kombination dieser Ansätze ist gleicherweise denkbar. Um eine willkürliche Festlegung von aktionsunabhängigen Kriterien zu vermeiden, müssten bereits Daten, bestenfalls aus der realen Welt oder einer Feldstudie, vorhanden sein. Ist der Transaktionsanstoß ausschließlich vom Nutzer abhängig, können Nutzerfehler oder beabsichtigte Manipulationen das Änderungsprotokoll unbrauchbar machen. Bei einem aktionsgetriebenen Transaktionsbeginn besteht die Gefahr, dass die Masse der zwischengespeicherten Daten die Transaktionskosten in einem Masse verteuert, wodurch diese für Miner unattraktiv wird oder die Transaktionen fehlschlagen lässt. Auf die *Dokument schließen*-Operation würde dies zutreffen. Externalisierungsoperationen, explizit *Speichern* und *Drucken*, haben dagegen den Vorteil, dass sie bequem von Nutzern ausgelöst werden können. Ebenfalls treten sie normalerweise häufiger auf. Die Verbreitung eines Dokuments ohne eine dieser Externalisierungsaktionen ist nicht möglich, wodurch sie sich bestens als Auslösepunkte eignen. Die Beschränkung auf den neusten Wert in einer geänderten Zelle ist in diesem Fall unproblematisch, da außer dem Nutzer niemand den Zwischenstand einsehen konnte. Insbesondere, da mehrmaliges Ändern einer Zelle häufig auf unbeabsichtigte Nutzeraktionen wie z. B. Tippfehler zurückzuführen ist.

3.3 Audit Trail Aufbau

Die benötigten Bestandteile eines Änderungsprotokolls können anhand der W-Fragen erschlossen werden. Dabei ist mindestens der Änderungszeitpunktes (wann?) eines Dokuments (wo?) mitsamt allen Änderungen (was?) nötig, um ein funktionelles Änderungsprotokoll zu erhalten. Durch Erweiterung des Editors (wer?) kann zudem die Einführung personeller Verantwortung ermöglicht werden. Weitere Informationen erscheinen nicht ausreichend relevant für den ersten Prototypen eines Änderungsprotokolls. Auch wenn nicht alle vorgeschlagenen Audit Trail-Informationen, z. B. wie oder warum die Änderung durchgeführt wurde, enthalten sind, haben diese Kriterien eine starke Ähnlichkeit mit dem Audit Trail von Nash et al. [23].

3.3.1 Metadaten

Die auf der Blockchain zu speichernden Daten bestehen, neben den offensichtlich enthaltenen Änderungen, aus Metadaten. Der Zeitpunkt kann aus der Blockchain-Transaktion ausgelesen werden, wodurch nur Informationen, welche zur Identifizierung von Dokument und Editor benötigt werden, zusätzlich beigefügt werden müssen. Diese sollten dabei möglichst eindeutig und manipulationssicher sein.

Editoridentifikator

Eine Identifizierung anhand des mit dem Tabellenprogramm verbundenen Benutzernamens ist zwar denkbar, jedoch bietet sich für diese Problemstellung eine Lösung mittels Ethereum-Accounts an. Weil eine Transaktion von einem Account angestoßen werden muss, ist dieser Lösungsansatz besonders intuitiv. Stellt eine Organisation jedem Nutzer mit Änderungszugriff einen eigenen Account bereit, so sind diese eindeutig identifizierbar. Eine Manipulation durch „fremde“ Accounts ist sofort offensichtlich und kann theoretisch auch im zu entwickelnden Smart Contract unterbunden werden. Manipulationsgefahr besteht folglich nur, wenn Nutzer ihre Account-Zugangsdaten nicht vertraulich behandeln. Der Nachteil dieses Ansatzes ist der administrative Aufwand, dass jederzeit ausreichend Guthaben auf allen Accounts vorhanden sein muss.

Dokumentidentifikator

Bieten Tabellenprogramme eine integrierte eindeutige ID zu jedem Dokument, sollte diese genutzt werden. Ist dies nicht der Fall, fallen weitere implizierte Dokumenteigenschaften, wie z. B. Speicherort, trotz ihrer lokalen Eindeutigkeit heraus, da diese sich über mehrere Systeme hinweg unterscheiden können. Dementsprechend bleibt die Verknüpfung mit einer generierten ID. Eine Scheineindeutigkeit nach dem Geburtstagsparadoxon [12], welches besagt, dass gegeben einem ausreichend großen Alphabet und einer langen Zeichenfolge die Chance von Duplikaten, selbst in großen Räumen gegen null geht, ist genauso denkbar wie die garantierte Eindeutigkeit basierend auf der Blockchain. Garantierte Eindeutigkeit kann mittels Schlüsselgenerierung unter Berücksichtigung der bereits existierenden Schlüssel oder durch die Verknüpfung mit einem neuen Ethereum-Account erreicht werden. Dabei sind die beiden letzteren Varianten nicht nur kostspielig, sondern erhöhen

auch die Kosten des geplanten Änderungsprotokolls durch die Internalisierung der Vorbedingungsgenerierung. Zweitere Variante missbraucht zudem Ethereum-Accounts für einen nicht vorgesehenen Anwendungsfall. Unter der Annahme, dass diese Accounts noch benutzt werden sollen, ist zudem eine Administration des damit verbundenen privaten Schlüssels oder Passworts erforderlich. Dementsprechend erscheint die Verwendung eines off-chain erstellten, praktisch eindeutigem Schlüssel als bessere Alternative.

3.3.2 Änderungen

Moderne Tabellen bestehen nicht nur aus reinen Daten, sondern sind auch häufig visuell formatiert. Beispiele dafür sind kolorierte Zellen oder versteckte Spalten. Trotzdem beschränkt sich dieser erste Prototyp nur auf Zellendaten. Diese sind Hauptbestandteil von Tabellen und sind damit auch der Schwerpunkt in einem Änderungsprotokoll. Eine Kostenexploration mit mehr als den funktionalen Mindestanforderungen erscheint zudem beschränkt aussagekräftig. Die angestrebte Minimalität des Änderungsprotokolls empfiehlt den Verzicht auf zusätzlichen Informationen, wie dem Wert vor der Änderung oder eine Änderungsbeschreibung. Ausnahme ist dabei das Hinzufügen und Löschen von Zeilen bzw. Spalten, da die Auflistung aller betroffenen Zellen nie kleiner als der Befehl selbst sein kann.

3.4 Einschränkungen

Die Anvisierung der öffentliche Blockchain bedeutet, dass alle Daten öffentlich eingesehen und gegebenenfalls manipuliert werden können. Im Falle von vertraulichen oder geschäftlich relevanten Daten ist dies problematisch. Die Daten werden jedoch in diesem umzusetzenden Prototyp noch nicht verschlüsselt. Eine einheitliche Verschlüsselung ist effektiv unnütz und auch eine individualisierte Variante bietet dem auf Umsetzbarkeit und Wirtschaftlichkeit fokussierten Prototypen keinen nennenswerten Vorteil. Auch eine Synchronisation über mehrere Nutzer ist mit lediglich diesen Bestandteilen nicht möglich. So könnten von einem alten Zustand aus entstandene Änderungen ohne Überprüfung neuere Zustände überschreiben oder zumindest verfälschen. Ansätze, die dieses Isolationsproblem lösen, sind denkbar, z. B. eine zusätzliche Variable, welche die aktuellste Version beschreibt, jedoch bedeuten mehr on-chain Operationen immer mehr Betriebskosten. Das heißt, dass jegliche Erweiterungen unweigerlich die Kosten in die Höhe treiben. Eine Atomarität von Externaliserungsoperation und Blockchain-Transaktion kann nicht garantiert werden, da eine erfolgreiche Transaktion nicht rückgängig gemacht werden kann. Ein Abbruch nach erfolgreicher Externalisierung, jedoch vor erfolgreichem Transaktionsende, ist denkbar. Gegeben der Wichtigkeit eines korrekten Änderungsprotokolls und der einfacheren Wiederherstellungsoption, sollte diese Transaktion bevorzugt behandelt werden. Dementsprechend kann dieser Ansatz lediglich Konsistenz und Dauerhaftigkeit der gewünschten ACID-Eigenschaft garantieren. Soll eine Einschränkung der Editoren eines Dokuments vorhanden sein, könnte dies nicht off-chain durchgeführt werden und hätte damit den zuvor genannten Kostenanstieg zur Folge. Eine Nutzerverwaltung ist zwingend nötig, damit die getroffene Entscheidung, der Verknüpfung von Editor mit Ethereum-Account, sinnvoll ist. Diese kann zwar off-chain passieren, da jedoch Guthaben nicht verlustfrei zwischen Accounts übertragen werden kann, ist auch diese nicht unproblematisch. Dazu kommt, die Gefahr des Missbrauchs des bereitgestellten Guthabens durch Editoren. Im schlimmsten Fall stiehlt der für die Administration verantwortliche Mitarbeiter Teile von dem des verfügbaren Guthabens.

3.5 Zusammenfassung

Die benötigten Komponenten und deren Anforderungen, um das Ziel eines manipulationssicheren Änderungsprotokolls von Tabellenanwendungen zu erreichen, wurden beschrieben. Dies ist in Abbildung 3.1 dargestellt. Einzig die Werteänderungen zwischen zwei Zuständen sollen automatisch erfasst werden. Ausgelöst durch ein lokales Externalisierungsereignis, können die gemachten Änderungen, minimiert auf den aktuellsten Zellenwert der betroffenen Bereiche, mit einer Smart Contract Transaktion auf der Ethereum-Blockchain gespeichert werden. Die Dokumentidentifikation soll über eine mit dem Dokument verbundene ID realisiert werden, welche nicht on-chain generiert wird, jedoch praktisch eindeutig ist. Editoren sollen über den verbundenen Ethereum-Account erkannt werden. Im folgenden Kapitel werden mögliche Smart Contract Versionen eines Tabellenänderungsprotokolls anhand der hier getroffenen Designentscheidungen exploriert.

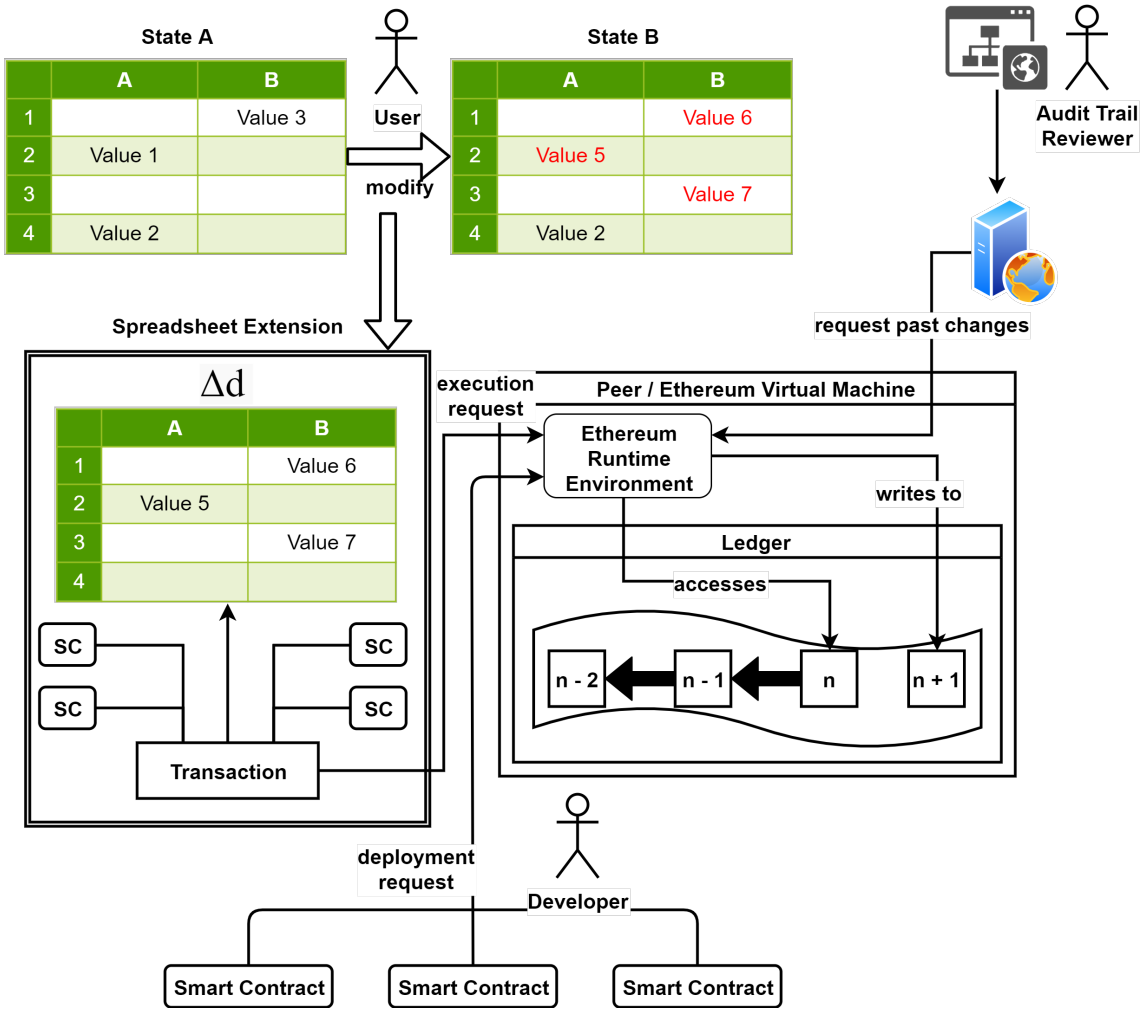


Abbildung 3.1: Konzeptionelle Komponenten eines Blockchain-basierten Änderungsprotokolls

Ein Nutzer nimmt Änderungen an einer Tabelle vor. Diese werden automatisiert zwischengespeichert. Durch ein Externalisierungsereignis werden die Änderungen in Transaktionen verpackt und an eine EVM übergeben. Darauf werden diese auf der Blockchain gespeichert und können von einem Webserver abgerufen werden, damit diese Änderungen für einen Audit Trail Reviewer auf einer Website angezeigt werden können.

4 Smart Contract Umsetzungen eines Tabellenänderungsprotokolls

Das beschriebene Konzept schreibt keine konkrete Speicherstruktur vor und erlaubt dadurch mehrere Umsetzungsvarianten. Vier mögliche Varianten werden in Solidity exploriert. Für eine werden weitere alternative Speichermöglichkeiten im Zuge der Optimierung untersucht. Die Einteilung in Varianten erfolgt unter dem Gesichtspunkt der Anzahl der gleichzeitig im World State gehaltenen Änderungen. Abschließend werden die erreichten theoretischen Ersparnisse diskutiert.

4.1 Variante 1: Gespaltene Transaktionen (Split Transactions)

Trotz der getroffenen Entscheidung für eine Massentransaktion ist die Exploration von Zelle-für-Zelle-Updates aus ökonomischer Sicht sinnvoll. Dies liegt an den momentan vergleichbaren Kosten für die Allokation von neuem Speicherplatz gegenüber dem Anstoßen einer neuen Transaktion verbunden mit der zweigeteilten Änderungsdefinition in Zelle und Inhalt. Aus den in Tabelle 2.1 auf Seite 22 aufgeführten Transaktionskosten ergeben sich Kosten von 21 000 *Gas* bei jedem Transaktionsbeginn und 5 000 *Gas* für jede Datenänderung nach erstmaligem initialisieren. Dies eliminiert sowohl die Möglichkeit für das Beschreiben von bisher ungenutztem Speicher 20 000 *Gas* bezahlen zu müssen, als auch die Gutschrift von 15 000 *Gas* für das Freigeben von Speicher. Mit diesen häufigen Transaktionen verkleinern sich allerdings Optimierungsmöglichkeiten der zu übertragenden Datenmenge und eine signifikante Kostenersparnis sollte zur Weiterverfolgung früh erkennbar sein. Während bei der anvisierten einmaligen Transaktion das Problem von theoretisch unendlich Kosten, gegeben einer entsprechend großen Änderungsmenge, und damit die Gefahr des Transaktionsabbruchs ohne resultierende Zwischenergebnisse besteht, kann in diesem Ansatz nach der zuletzt verarbeiteten Änderungen fortgesetzt werden. Damit ist das Änderungsprotokoll zwar nicht direkt vollständig, aber lediglich die zeitlich späteren Änderungen müssen ergänzt werden, was ohne eine zusätzliche Aufteilung erfolgen kann. In der realen Welt ist die Nutzbarkeit allerdings fraglich, da aufgrund der nicht garantieren Reihenfolge von Transaktionen, die nächste erst angestoßen werden kann, sobald die vorherige erfolgreich im neusten Block enthalten ist. Gegeben der ungefähr 15 sekündigen Blockdauer bei Ethereum, kann ein komplettes Update lange dauern.

Neben den in Abschnitt 3.3.1 beschriebenen Metadaten *Editor* und *Dokumentidentifikator* wird jeweils nur ein Attribut für Zellenidentifikation und Zelleninhalt benötigt. Dies ist in Listing 4.1 dargestellt. Dabei existieren zwei unabhängige Varianten, um die Kostenunterschiede zwischen dem String- und Bytes-Format zu quantifizieren. Die 20 Bytes Account-Adresse des Editors ist die erste systemdefinierte Größe. Da die EVM in 32 Bytes Schritten arbeitet, verbleiben 12 Bytes, welche jedoch zu wenig sind um eine praktisch eindeutige Identifizierung zu gewährleisten. Allerdings könnte sich dies für Zellenidentifikation oder Zelleninhalt als ausreichend erweisen und

Listing 4.1 Attribute der beiden Umsetzungen von Variante 1

```
contract SplitTransactionsStringP {
  bytes32 internal fileId;
  address internal editor;
  string internal cell;
  string internal content;
```

```
contract SplitTransactionsBytes {
  bytes32 internal fileId;
  address internal editor;
  bytes internal cell;
  bytes internal content;
```

die Zahl der benötigten Speicherplätze kann bei einer gruppierten Deklaration reduziert werden. Dementsprechend kann dem Dokumentidentifikator ein voller Speicherplatz zur Verfügung gestellt werden, welcher aus Kostengründen mit einer festen Größe deklariert wird.

4.2 Variante 2: Globale Kopie (Global Copy)

Ein möglicher Ansatz, um die gesammelten Datenänderungen permanent auf der Blockchain zu speichern, ist diese direkt für die Rekonstruktion der Tabelle auf der Blockchain zu verwenden. Auf diese Weise kann der aktuelle Stand jedes Dokuments schnell und problemlos von der Blockchain ausgelesen werden. Der mit dieser Variante verbundene Speicherbedarf und die Komplexität der Update-Operation sind offensichtlich deutlich höher als in der vorausgegangenen Variante. Infolge dieser Internalisierung sind die erwarteten Kosten verhältnismäßig hoch. Dafür erlaubt ausschließlich diese Variante den direkten Zugriff auf alle Änderungen, ohne die vorherigen Blöcke besuchen zu müssen. Dabei ist anzunehmen, dass das Updaten vorhandener Zellen effizienter ist als eine Liste mit neuen Änderungen potenziell unendlich zu erweitern. Dies ist zutreffend, sobald vorhandene Werte regelmäßig geändert werden und die Update-Operation nicht die Kosten von einer Speicherallokation überschreitet.

Listing 4.2 zeigt die für die Umsetzung benötigten Attribute. Solidity-Einschränkungen sind dabei schnell zu erkennen. Neben der Map, welche für die Assoziation zwischen Dokumentidentifikator und Dokument dient, wird ein zusätzliches Array für die IDs benötigt, da es keine Möglichkeit gibt auf die Schlüsselmenge einer Map zuzugreifen. Ohne dieses Set ist nicht einsehbar, für welche Dokumente eine Repräsentation auf der Blockchain vorhanden ist. Die kostengünstigere Lösung mittels eines zweiten Arrays für die Dokumente, welche über den gleichen Array-Index verbunden sind, würde zwar das Speichern der ID in mehreren Datenstrukturen verhindern, aber nicht die Eindeutigkeit von Dokumenten garantieren. Gleichzeitig ist die Zugriffszeit auf jedes Dokument in $O(1)$ nicht mehr garantiert und damit würden sich Kosten anderweitig erhöhen. Allerdings ist für das Mapping ein zusätzlicher Boolean nötig, um erkennen zu können, dass der Eintrag zu einem Schlüssel tatsächlich vorhanden ist. Das gleiche Prinzip gilt für das Mapping zwischen Zellenbeschreibung und Zelleninhalt. Um Zellen effizient entfernen zu können, wird zusätzlich noch die Position des als

Listing 4.2 Attribute der Umsetzung von Variante 2

```
contract GlobalCopy {
  struct File {
    bytes32[] cellIds;
    bool isFile;
    address lastEditor;
    mapping(bytes32 => Cell) cells;
  }
  struct Cell {
    bool isCell;
    uint248 cellIdIndex;
    bytes32 content;
  }
  bytes32[] internal fileIds;
  mapping(bytes32 => File) internal files;
```

Schlüsselmenge verwendeten Arrays benötigt. Auf eine weitere Unterteilung der Zellenidentifikation in Tabellenblätter mit zugehöriger purer Zeilen- und Spaltenbeschreibung, welche gegebenenfalls weiter encodiert werden kann, wird verzichtet. Trotz Verkürzung jeder Zellenbeschreibung um den Tabellennamen, ist kein eingesparter Speicherplatz, aufgrund der benötigten Speicherstrukturredundanz, zu erwarten. Zudem wäre diese Aufteilung kostspielig, da sie auf der EVM ausgeführt werden müsste, weil selbst definierte Typen in der Methodensignatur momentan noch ein experimentelles Feature sind, welches Fehler verursachen kann. Dabei basiert diese Entscheidung auf der Annahme, dass das Löschen von Tabellenblättern und deren Umbenennung selten auftritt. Den diese erfordern eine Iteration der kompletten Liste. Die Reduktion der Zellbeschreibung und des Zellinhalts auf eine fest definierte Länge ist auf dieselbe Solidity-Einschränkung zurückzuführen. Genauer wird *Bytes* durch ein *Bytes1* Array dynamischer Größe beschrieben und verschachtelte Arrays sind noch nicht standardmäßig möglich. Dies hat den positiven Nebeneffekt des verringerten maximalen Speicherbedarfs, resultiert aber zugleich in abgeschnittenen Daten, falls diese zu lang sind. Letztlich muss angemerkt werden, dass die in Kapitel 9 geäußerte Annahme, dass ein Vermerk mit den geänderten Zeilen oder Spalten, im Vergleich zu allen betroffenen Zellen, kostengünstiger zu speichern ist, nicht für diese Variante zutreffend ist. Dies ist dem Umstand geschuldet, dass die einzelnen gespeicherten Zellen geupdatet werden müssen. Aufgrund fehlender Bereichszugriffsfunktion erfordert dies die Iteration über alle vorhandenen Zellen.

4.3 Variante 3: Massentransaktion (Bulk Transaction)

Der Mittelweg der beiden vorangegangenen Varianten ist das Speichern von ausschließlich dem letzten Änderungsprotokoll. Folglich sind die Blockchain-Eigenschaften, wie in Variante 1, lediglich für die gemachten Änderungen, allerdings nicht für die Tabellenrekonstruktion, garantiert. Dies reduziert zwar die benötigten Kosten, allerdings müssen nun alle vorherigen Blöcke und deren Transaktionen erneut überprüft werden.

Listing 4.3 Attribute der Umsetzung von Variante 3

```
contract BulkTransaction {
  bytes32 internal fileId;
  address internal editor;
  bytes32[] internal cells;
  bytes32[] internal contents;
```

Listing 4.4 Attribute der teilweise encodieren Optimierung von Variante 3

```
contract BulkTransactionPartlyEncoded {
  bytes32 internal fileId;
  address internal editor;
  bytes internal changes;
```

Die vorangegangenen Solidity-Einschränkungen können auch in der in Listing 4.3 dargestellten Variante nicht komplett umgangen werden, jedoch ist hier nur die Einhaltung der Änderungsreihenfolge, aber ohne Zugriff via Id, gefordert. Deshalb sind zwei Arrays eine korrekte und zugleich die kosteneffizienteste Lösung. Resultierend aus der fehlenden Möglichkeit, den benötigten Speicherplatz dynamisch zu allokieren, entsteht ein deutlicher Overhead. So benötigten sowohl eine einstellige Ganzzahl, als auch ein 30 Zeichen langer String in adjazenten Zellen zwei volle 32 Bytes Slots.

4.4 Optimierte Speichermöglichkeiten für Variante 3

In einer vorläufigen Kostenerprobung hat sich der Ansatz der Massentransaktion als kompetitiv erwiesen. Da diese Speicherstruktur die meisten Optimierungsmöglichkeiten bietet, werden verschiedene Ansätze iterativ exploriert. Zuerst werden serialisierte Speicherformen untersucht, um die gewonnenen Erkenntnisse dann in der letzten Variante an einem alternativen Speicherort zu erkunden.

4.4.1 Teilweise Encodiert

Das Ziel der, in Listing 4.4 gezeigten, Optimierungsvariante ist den Overhead, durch die Ersetzung der beiden kostspieligen Arrays mit einem Attribut dynamischer Länge, zu reduzieren. Gleichzeitig kann dies die vorangegangene Einschränkung auf maximal 32 Zeichen aufheben. Dabei werden die zu speichernden Änderungen in einem unabhängigen Format encodiert und dann in dieses neue Attribut geschrieben. Externe Formate sollten platzsparend sein und, im Gegensatz zu Solidity, dynamische Längen ohne anschließendes Auffüllen unterstützen.

Listing 4.5 Minimiertes ProtoBuf Format für Variante 3

```
syntax = "proto3";
message BCSSContractData {
  repeated Sheet sheets = 1;
}
message Sheet {
  string name = 1;
  repeated MapFieldEntry changes = 2
}
message MapFieldEntry {
  string cell = 1;
  string content = 2;
}

```

JSON

Ein weitverbreitetes und sehr flexibles Format ist JSON. Sprachenunabhängigkeit und die einfach lesbare textbasierte Syntax sind einige der mit diesem Format verbundenen Vorteile. Auch die Aufteilung in Attribut-Wert-Paare ist im Kontext der zu speichernden Daten passend. Die Anzahl der minimal zusätzlichen Zeichen für x Änderungen sind durch $\begin{cases} 2, & \text{if } x = 0 \\ 2 + 5 + 6 * (x - 1), & \text{if } x \geq 1 \end{cases}$ begrenzt. Allerdings sind weitere Einsparungsmöglichkeiten bei der Extraktion der Tabellenblätter limitiert. Die ausgeschriebenen Attributbezeichnern würden viel Platz in Anspruch nehmen.

Protocol Buffers

Dieses von Google entwickelte Serialisierungsformat[1] ist ein Binärformat und sowohl unter dem Gesichtspunkt der effizienten Kodierung als auch einen minimalen Overhead entwickelt worden. ProtoBuf benötigt keine expliziten Trennzeichen durch die Angabe eines indexierten Datenformates und der Anzahl der zugehörigen Werte-Bytes. Im Vergleich zu JSON erhöht sich die Platzersparnis mit steigender Schemakomplexität. Eine Vielzahl von Sprachen, darunter die potenziell verwendeten, werden unterstützt. Es existieren zwei Schemas, eins das äquivalent zur JSON Encodierung ist und eine optimierte Variante. Listing 4.5 zeigt dieses optimierte Schema. Auf eine Liste mit Pseudomapeinträgen muss zurückgegriffen werden, da die native Implementierung von Maps in ProtoBuf die Elemente nach ihrem Schlüssel sortiert. Allerdings wird die Beibehaltung der Einfügereihenfolge benötigt. Ein weniger optimiertes Pendant zu dem verwendeten JSON-Schema existiert auch. Dieses besteht ausschließlich aus einer Liste für *MapFieldEntry*.

[1] https://developers.google.com/protocol-buffers

Listing 4.6 Attribute der Event-Optimierung von Variante 3

```
contract BulkTransactionEvent {
  event ChangedCells(bytes32 indexed fileId, bytes changes);
```

4.4.2 Vollständig Encodiert

Um zu garantieren, dass eine komplette Externalisierung aller Daten nicht effizienter ist, wird ein weiterer Smart Contract entwickelt. Dieser besteht nur aus einem einzigen *Bytes*-Attribut. Dafür werden die *fileId* und *editor* Attribute in das externe Serialisierungsformat aufgenommen. Explizit wird dafür ein weiteres ProtoBuf-Schema genutzt. Dieses wird dann in das Bytes Feld geschrieben.

4.5 Variante 4: Ethereum-Event (Bulk Event Transaction)

Alle bisherigen Daten wurden im *World State* der EVM gespeichert. Normalerweise ist dies die empfohlene Speicherungsweise, da nur daraus sofortiger Zugriff auf den aktuellen Stand möglich ist. Da der aktuelle Stand in Variante 3, genauso wie in Variante 1, jedoch nicht vom vorherigen Stand abhängig ist, müssen für die Tabellenrekonstruktion ohnehin vorherige Blöcke besucht werden. Aus diesem Grund kann das Speichern im Eventlog erwogen werden. Dies ist mit wesentlich verringerten Kosten verbunden. In dieser Umsetzung sind keine Attribute notwendig. Benötigte Felder werden als Parameter dem Event übergeben. Aus der in Listing 4.6 dargestellten Strukturierung ist erkennbar, dass sich für die teilweise encodierte Speichermethode aus Abschnitt 4.4.1 entschieden wurde. Die Daten werden mittels des ProtoBuf-Schemas aus Listing 4.5 serialisiert. Durch die Indexierung der Dokumentidentifikator, kann explizit nach diesem gefiltert werden. Der Editor muss nicht angegeben werden, da der Account durch dessen Transaktion ein Event ausgelöst wurde, bereits im gespeicherten Log enthalten ist. Dies erhöht die Kosten nur marginal, bietet allerdings einen deutlichen Mehrwert. Aus Tabelle 2.1 auf Seite 22 kann erkannt werden, dass sich die Kosten mit Indexierung auf $375+375+changesByteLength*8$ und ohne Indexierung auf $375+32*8+changesByteLength*8$ belaufen. Dementsprechend könnten weitere 119 *Gas* potenziell eingespart werden. Basierend auf dem in Abschnitt 2.2.1 auf Seite 17 genannten durchschnittlichen *Gas*-Preis und Ethereum-Preis ergibt dies eine Ersparnis von 0,000412 US$.

4.6 Diskussion und Zusammenfassung

Es wurden vier Grundspeichervarianten vorgestellt. Die erste aktualisiert eine Zelle pro Transaktion und zeichnet sich durch voraussichtlich konstante Kosten und Zwischenergebnisse aus. Während Variante 2 die komplette Tabelle im World State der Blockchain speichert. Dies hat den Vorteil, dass die aktuellsten Daten jederzeit abgerufen werden können, ist jedoch voraussichtlich auch mit den höchsten Kosten verbunden. Variante 3 speichert nur die neusten gesammelten Änderungen. Dieser Ansatz bietet von den bisher genannten Umsetzungen die meisten Optimierungsmöglichkeiten und wurde deshalb für eine weitere Kostenreduktion ausgewählt. Eine Speicherplatzreduktion von ungefähr Faktor fünf konnte dabei erreicht werden, wie in Tabelle 4.1 zu erkennen ist. Der benötigte

Nativer Ansatz	JSON	ProtoBuf	Minimized ProtoBuf
cells[] = { $Tabelle1.$A$5, $Tabelle1.$C$13, $Tabelle1.$J$42 } contents[] = {Hallo, 3, 150 }	{"$Tabel-le1.$A$5":"Hallo", "$Tabel-le1.C13":"3", "$Tabel-le1.$J$42":"150"}	0a170a0e5461 62656c6c65314135 1205486 16c6c6f 0a140a0f546162 656c6c6531433133 1201330a160a0f 546162656c6c6531 4a34321203313530	0a2d0a08546162 656c6c6531120b0a 02413512054861 6c6c6f12080a03 433133120133 12 0a0a034a3432 1203313530
256 (32 * 6) Bytes	75 Bytes	71 Bytes	47 Bytes

Tabelle 4.1: Speichergrößen der Variante 3 Versionen

Speicherplatz ist für die gleichen Werte dargestellt. Die größte Einsparung kann von der nativen Umsetzung zu einer teilweise encodierten festgestellt werden. Ein anderes Encodierungsformat führt zu einer verhältnismäßig kleineren Einsparung. Allerdings kann mit einer weiteren Optimierung der benötigte Platz um weitere 33,8% reduziert werden. Die bisherigen Ergebnisse wurden wurden abschließend für eine Variante benutzt, welche in der Log-Bereich der Blockchain anstelle des World State speichert. Das folgende Kapitel vergleicht existierende Tabellenanwendungen um eine informierte Auswahl für die kommende Implementierung treffen zu können.

5 Vergleich von existierenden Tabellenanwendungen

Nach der Konzeption eines Änderungsprotokolls von Tabellen muss eine entsprechende Anwendung ausgewählt werden. Drei existierende Tabellenanwendungen werden in diesem Kapitel auf ihre Umsetzungskompatibilität untersucht. Diese Vorauswahl der zu betrachtenden Tabellenprogramme wurde anhand von drei Kriterien getroffen. Am stärksten gewichtet ist dabei die Existenz und die verbundene Dokumentation einer Schnittstellenanbindung für Drittentwickler. Sekundäre Entscheidungsmerkmale sind die noch fortlaufende Entwicklung beziehungsweise der andauernde Support einer Anwendung, sowie dessen Relevanz basierend auf der Anzahl der Nutzer. Dies führte zu einer Untersuchung von *Microsoft Excel*, *LibreOffice Calc* und *Google Sheets*. Vor deren genaueren Betrachtung werden zuerst generelle Bewertungskriterien der Untersuchung definiert. Globale aber fachlich irrelevante Unterschiede sind nicht Teil dieser Bewertung. Begrifflichkeiten orientieren sich nach der betrachteten Anwendung, woraus unterschiedliche Betitelung ähnlicher Funktionen, sowie identische Benamung für andersartige Features folgen. Darauf folgt die Untersuchung der drei Programme. Abschließend werden die gewonnenen Erkenntnisse in dem für die Anwendungsauswahl entscheidenden Vergleich diskutiert.

5.1 Bewertungskriterien

Die zu betrachteten Aspekte werden zunächst in der Auftrittsreihenfolge für Nutzer aufgelistet. Basierend auf dem in Kapitel 3 beschriebenen Konzept, ist dies Installation, Aktivierung eines Dokuments, Behandeln eines Änderungsereignis und abschließend Auftreten eines Externalisierungsereignisses.

Flexibilität in der Einbindung der zu entwickelnden Komponente, umfasst sowohl die Anzahl differenzierter Ansätze, z. B. mögliche Programmiersprachen, als auch die benötigten Schritte zur Erweiterung. Der Erweiterungsaufwand für einzelne bzw. mehrere Dokumente ist dabei getrennt zu betrachten, jedoch mit einem höheren Fokus für mehrere Dokumente. Existiert eine eindeutige Dokumentidentifikator, wirkt sich marginal positiv aus.

Aktivierungsaufwand Bestehende Möglichkeiten des simultanen Starts mit der Tabellenanwendung sind einer manuellen Aktivierung vorzuziehen. Optimal ist dieses Kriterium erfüllt, sobald Anweisungen direkt und gezielt ausgeführt werden können.

Änderungsereignisse Das Vorhandensein, die Auslösung und Behandlung sind Teil dieses Gesichtspunktes. Dabei ist eine minimale genaue Benachrichtigungsweise, z. B. durch ein Ereignis pro Änderung, allgemeineren Hinweisen, z. B. in Form einer Flagge, die vorhandene Änderungen anzeigt, zu bevorzugen. Möglichkeiten die vorhandenen Strukturen zu umgehen, also die Manipulationssicherheit, ist mit inbegriffen.

Kontrolle über Externalisierungsereignisse Dies umfasst sowohl die Möglichkeit Aufrufe der Speicher- und Druckfunktion mitzubekommen, abzufangen und eigenständig aufzurufen. Aufgrund der gewünschten nutzergetriebenen Externalisierung, ist eine dauerhafte Synchronisation, wie in Online-Varianten gegeben, als eher negativ zu bewerten.

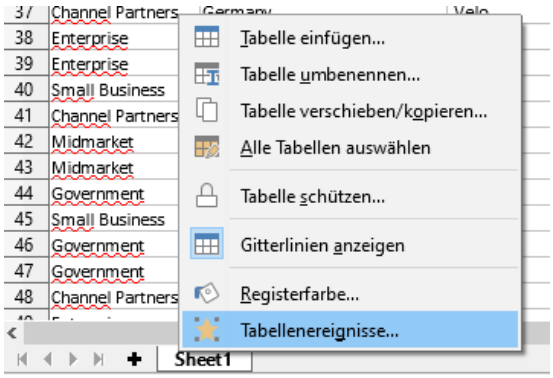
In absteigender Entscheidungsrelevanz aufgelistet, sind die möglichen Behandlungsweisen von Änderungsereignissen, das wichtigste Kriterium. Darauf folgt die Kontrolle über Externalisierungsereignisse. Da ohne diese beiden Aspekte das anvisierte Konzept nicht in dieser Art umgesetzt werden kann. Der Funktionalität folgt das Ziel den zusätzlichen Aufwand für Nutzer gering zu halten. Dementsprechend haben die komfortable Einbindung und Aktivierung eine geringere Priorität. Wobei der Einbindung eine leicht höhere Entscheidungskraft zugeschrieben wird, um das benötigte technische Wissen von Nutzern nicht zusätzlich zu beanspruchen.

5.2 Microsoft Excel

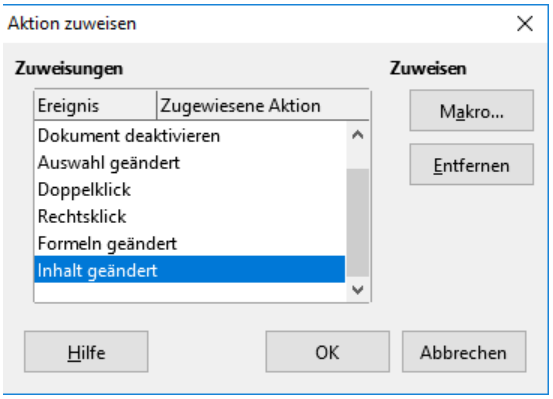
In Microsoft Office enthalten und das bekannteste Tabellenprogramm ist *Microsoft Excel*. Einzelne Dokumente können über Makros, geschrieben in Visual Basic For Applications (VBA), erweitert werden. In Änderungsevents können dort über den reservierten Funktionsnamen *Worksheet_Change* umgesetzt werden. Makros können allerdings auch installiert werden und damit in allen Dokumenten verfügbar sein. Installierbare Add-Ins können mittels des Microsoft .NET Frameworks und damit unter anderem in C# entwickelt werden. Die zugrunde liegende API ist dabei identisch. Diese sind allerdings alle nur für die Windows Desktopversion verfügbar. Sogenannte *Office Add-Ins* werden in JavaScript programmiert und können auch in anderen Umgebungen, einschließlich der Webvariante, verwendet werden. Unterschiede zwischen dieser und der .NET API sind vorhanden.So bietet beispielsweise nur die .NET API Ereignisse für *Speichern*, *Drucken* und *Exportieren* an. Funktionen können proaktiv und reaktiv ausgeführt werden. Unabhängig der verwendeten API kann die Speicherfunktion aufgerufen werden und auch das *Erstellen*, *Öffnen* und *Schließen* von Dokumenten kann mittels designierter Ereignisse bemerkt werden. Änderungslistener können auf einzelne oder alle Tabellenblätter oder auf fest definierte Teilbereiche registriert werden. Diese enthalten dabei einen Verweis auf das geänderte Tabellenblatt mitsamt des geänderten Bereichs, dem Änderungsgrund und im Falle von exakt einer geänderten Zelle den jetzigen und vorherigen Wert.Änderungsgründe umfassen das Einfügen oder Löschen von Zellen, Zeilen und Spalten sowie die Änderung eines Bereichs. Änderungen die durch die *Sortieren*-Funktion entstehen oder an Tabellenblättern, werden jedoch nicht erfasst. Zudem besteht die Möglichkeit alle Events durch Ändern eines Wertes zu deaktivieren. *Microsoft Excel*besitzt eine Funktionalität mit der alle Änderungen aufgezeichnet werden können, jedoch kann auf die nicht per API zugegriffen werden.

5.3 LibreOffice Calc

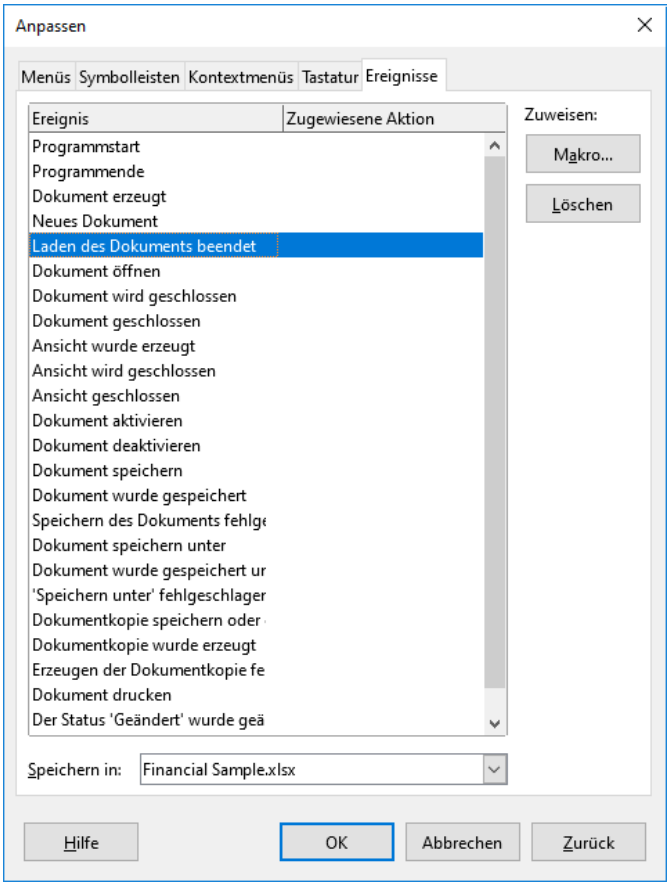
Nach der Abspaltung von OpenOffice und der unabhängigen Weiterentwicklung ist *LibreOffice Calc* weiterhin als Open Source Software verfügbar. Es ist damit die einzige Anwendung, welche uneingeschränkt nach dem gegebenen Bedarf erweitert werden kann. Da die Betrachtung wegen der Vergleichbarkeit allerdings aus Sicht eines Drittentwicklers durchgeführt wird, würde diese Alternative erst nach der Feststellung einer unzufriedenstellenden Erweiterbarkeit aller Anwendungen



(a) Menüpunkt Tabellenereignisse



(b) Tabellenereignisse



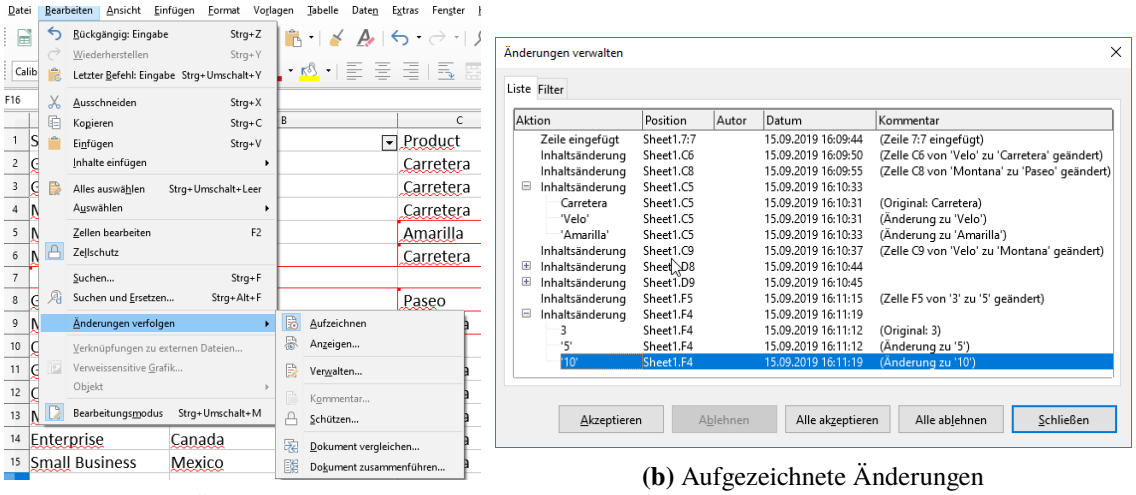
(c) Dokumentereignisse

Abbildung 5.1: *LibreOffice Calc* Ereignisse mit UI-Verbindungsmöglichkeiten für Makros

Manuelle Verknüpfung eines Makros mit Änderungsereignissen

in Betracht gezogen werden. Die vorhandene API bietet zwei Möglichkeiten der Programmerweiterung. Makros können Dokument- sowie Systemlokal verfügbar zu sein und unterstützten die Programmiersprachen LibreOffice Basic (basierend auf StarOffice Basic), BeanShell, JavaScript und Python. Erweiterungen können alternativ in C++ und Java programmiert werden. Dabei ist die verfügbare unterliegende API für alle Sprachen nahezu identisch mit Ausnahme einiger Komfortfunktion, wie z. B. das erleichterte Hinzufügen von Menüs in Erweiterungen. Makros können Makrofunktionen anderer Makros und von Erweiterungen bereitgestellte Funktionalitäten aufrufen. Das Gleiche gilt und auch für Erweiterungen. Funktionen werden, unabhängig der gewählten Umsetzungsform, nur durch explizites Aufrufen ausgeführt, wofür sowohl eine Vielzahl an Ereignissen und UI-Erweiterungsmöglichkeiten bereitgestellt werden. *LibreOffice Calc* bietet zudem die Möglichkeit, Makros über die Benutzeroberfläche, beispielhaft dargestellt in Abbildung 5.1, an verschiedene Ereignisse zu binden. Im Falle der in Abbildung 5.1b dargestellten Änderungsereignissen lösen diese allerdings nur bei direkten Zelldatenänderungen über die Benutzerschnittstelle aus und müssen auch jedem Blatt einzeln hinzugefügt werden. Die API bietet zwei verschiedene Änderungslistener. Mittels des *XChangesListener* kann die Art der Änderung, z. B. Zeilenmanipulation oder Datenänderung, festgestellt werden. Zudem ist ausschließlich der Zugriff auf den geänderten Bereich möglich. Es existieren Änderungsoperationen, z. B. Änderungen via Makro oder das Sortieren eines Bereichs, welche kein Ereignis auslösen. Der *XModifyListener* erlaubt dagegen, weder die Feststellung der Änderungsart, noch ist der geänderte Bereich nachvollziehbar. Beide Listener müssen nicht auf Dokumentebene registriert werden, so kann der *XModifyListener* beispielsweise auch einzelnen Zellen angefügt werden, um den Änderungsbereich einzugrenzen. Auch lösen Ereignisse, welche weder Daten noch die Zellstruktur ändern, wie beispielsweise Schriftgrößenanpassungen, die zugehörigen Broadcaster aus. Änderungen der Tabellenblätter können über den *XModifyListener*, aber nicht den *XChangesListener* festgestellt werden. *LibreOffice Calc* besitzt eine funktionale Änderungshistorie mittels des in Abbildung 5.2a gezeigten Menüpunkts *Änderungen aufzeichnen*. Diese Funktionalität unterstützt ähnlich dem *XChangesListener*allerdings auch nicht alle Änderungsereignisse. Die aufgeführte Änderungshistorie ist, wie in Abbildung 5.2b zu sehen ist, sehr ausführlich. Zusätzlich stellt sie gemachte Änderungen auch durch eine rote Zellumrandung in der Tabelle dar (vergleiche Abbildung 5.2a). Zum Zeitpunkt der Untersuchung kann diese zum Versionsmanagement gedachte Funktionalität allerdings nur mit verfügbaren API zwischen aktiviert und deaktiviert umgeschaltet werden. Gegeben, dass auf die äquivalente Funktionalität im für Textdokumente verwendeten Unterprogramm der *LibreOffice Suite* vollständig zugegriffen werden kann, ist eine zukünftige Erweiterung der vorhandenen Schnittstelle denkbar und zumindest möglich.

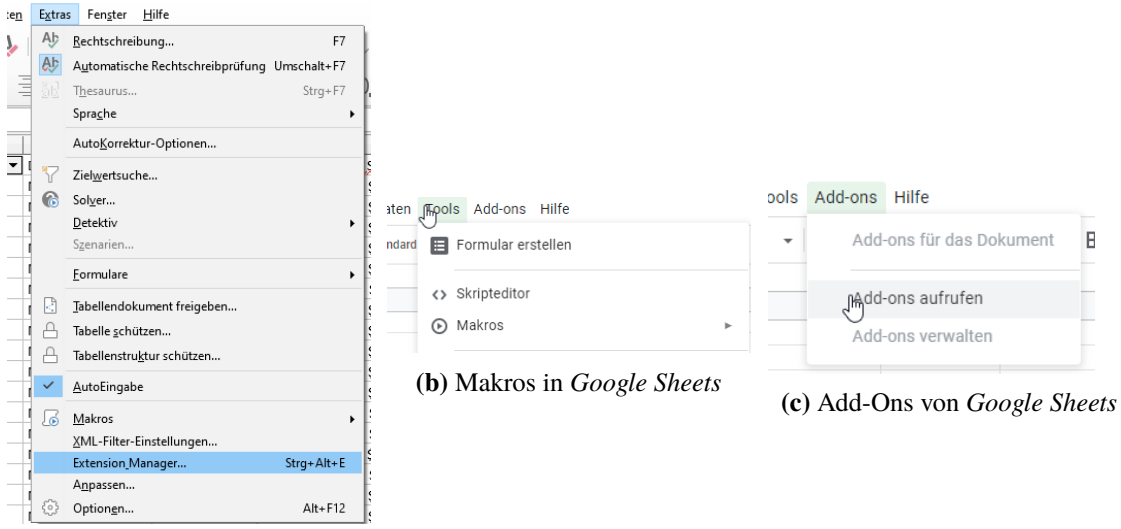
Externalisierungsaktionen können mittels der API auf zwei unterschiedliche Arten angestoßen werden. Erstere Möglichkeit ist durch Nutzung der spezialisierten Schnittstellen *XStorable* bzw. *XPrintable* erreichbar. Der andere mögliche Ansatz ist mithilfe der *Dispatch Commands*, welche jede potenzielle Aktion umfassen. Dabei ist neben dem Anstoß einzelner Aktionen, auch deren beliebige Erweiterung erdenklich. Eine der Zusätzlich erlaubt *LibreOffice Calc* auch eine reaktive Vorgehensweise und bietet Dokumentlistener an, welche vor dem Anstoß der *Speichern-* bzw. *Drucken*-Operation oder nach erfolgreichem Speichern ausgelöst werden. Benutzerdefinierte Eigenschaften einzelner Dokumente können über die API gesetzt werden. Installierte Erweiterungen können in über das in Abbildung 5.3a gezeigte Menü aufgerufen werden. Darin können diese mit einem einzigen Klick deaktiviert werden.



(a) Menüpunkt Änderungen aufzeichnen

(b) Aufgezeichnete Änderungen

Abbildung 5.2: Natives Änderungsprotokoll von *LibreOffice Calc*



(a) Erweiterungsmenü von *LibreOffice Calc*

(b) Makros in *Google Sheets*

(c) Add-Ons von *Google Sheets*

Abbildung 5.3: Erweiterungsmöglichkeiten von *LibreOffice Calc* und *Google Sheets*

5.4 Google Sheets

Die von Google angebotene Software-As-A-Service Anwendung *Google Sheets* unterscheidet sich von den vorangegangen Tabellenprogrammen darin, dass sie von Nutzern normalerweise in einem Webbrowser bedient wird. Das Dokument wird auf einem Google Server gespeichert und mehrere Nutzer können es simultan bearbeiten. Dies hat zwei relevante Nebeneffekte, erstens ist jedes Dokument mittels eines eindeutigen URL-Pfads identifizierbar und zweitens existiert keine Möglichkeit ein Dokument explizit zu speichern. Die Erweiterung ist im betrachteten Kontext einerseits gebunden an die Webanwendung und andererseits als externe Anwendung möglich. Erstere Variante

basiert auf dem *Google Apps Script* Framework und wird auf Googles Infrastruktur ausgeführt. Es bietet sogenannte *Trigger*, welche im Falle limitiere Ereignisse wie einer Datenänderung automatisch ausgeführt werden. Konkret wird dies mittels reservierten Funktionsnamen umgesetzt. Dabei ist eine weitere Unterscheidung in ein einfaches Script oder als installiertes Add-On nötig. Im Falle eines Scripts, enthält das Änderungsereignis einen Verweis auf das geänderte Dokument, den verantwortlichen Google Nutzer Account, den geänderten Bereich und im Falle von lediglich einer veränderten Zelle den alten und neuen Wert. Scripte müssen jedem Dokument einzeln hinzugefügt werden und sind auf JavaScript und die existierende Framework-Funktionen limitiert. Dementsprechend wäre zum Anstoß einer Ethereum-Transaktion entweder ein weiterer Webservice, z. B. in Form einer Serverless Funktion, oder die Nutzung einer existierender REST-API nötig. Auch wird dieses Ereignis nur für Datenwertänderungen ausgelöst, womit Zeilen- und Spaltenänderungen nicht abgedeckt sind. Ein installierbares Add-On kann hingegen benutzerweit verbreitet werden. Dieses Änderungsereignis kann zusätzlich auch Zeilen- bzw. Spaltenmanipulationen widerspiegeln und wird daneben um eine Enum-Beschreibung erweitert. Außerdem kann die Entwicklung in Go, Java, Node.js, PHP, Python und Ruby stattfinden, wodurch Ethereum-Transaktionen direkt eingebunden werden können. Eine Verbreitung von Script, sowie Add-Ons ist allerdings auch automatisiert über ein weiteres Script und die REST-API von *Google Apps Script* möglich. Das Hinzufügen und entfernen von Makros und Add-Ons wird über die in Abbildung 5.3b respektive Abbildung 5.3c präsentierten Menüs ausgeführt.

Bei einer Implementierung als externe Anwendung wird zugunsten eines Speicherereignisses auf Änderungsereignisse verzichtet. Trotz des Fehlens einer expliziten nutzergesteuerten Speicheraktion stellt Google verschiedene Dokumentversionen bereit, wobei die Änderungen zwischen Versionen von mehreren Nutzern stammen können. Dies kann dazu genutzt werden, mittels der *Google Drive API* auf eine neue Revision zu reagieren und dann mit der *Google Sheets API* die Änderungen zwischen beiden Versionen zu bestimmen. Eine Umsetzung kann in Java, Node.js und Python erfolgen, erfordert allerdings den dauerhaften Betrieb der entwickelten Anwendung. Auch das Hinzufügen neuer Dokumente und Nutzer ist mit einem beträchtlichen Mehraufwand versehen, da gültige Credentials benötigt werden.

5.5 Diskussion und Auswahl

Von den betrachteten drei Anwendungen wird *Google Sheets* als erster Kandidat ausgeschlossen, obwohl es die einzige Anwendung mit einem explizit eindeutigen Dokumentidentifikator ist. Gegeben der Online-Funktionalität ist es nicht offensichtlich, wie die anvisierte nutzergetriebene Externalisierung in *Google Sheets* umgesetzt werden kann. Der von Google erstellte Versionsverlauf, könnte gegebenenfalls dazu dienen, Speicherpunkte zu erhalten, allerdings entsprechen diese nicht explizit von dem Nutzer ausgelösten Aktionen. Da auf diesen Teil der Schnittstelle nicht innerhalb der Anwendung zugegriffen werden kann, müsste das zu entwickelnde Modul permanent und unabhängig von der Tabellenanwendung betrieben werden. Dies würde nicht der gewünschten Erweiterung entsprechen. Es würden zudem nicht Änderungsereignisse behandelt, sondern die Änderungen zwischen Versionen nachträglich ermittelt. Die Initialisierung neuer Dokumente und Autorisierung neuer Nutzer ist wegen der benötigten Google-Account-Credentials zudem verhältnismäßig aufwendig. Auch reduziert diese Abhängigkeit von einem externen Modul die Kohäsion zwischen Tabellenanwendung und zu entwickelnder Erweiterung. Bei der Nutzung der Änderungsereignisse wiederum, müsste jede Änderung direkt auf die Blockchain geschrieben werden, da *Schließen-*

und *Drucken*-Operationen nicht abgefangen werden können. Eine Erweiterung mittels installiertem Add-On ist der Script-Variante in allen Punkten überlegen. Beide Varianten bieten den Vorteil, dass außerhalb der erstmaligen Initialisierung eines Dokuments, keine weiteren Verantwortungen auf den Nutzer zukommen. Erstere Variante würde allerdings den Aufwand des dauerhaften Betriebs der externen Komponente voraussetzen. Im Ganzen würde eine Umsetzung des Konzepts in *Google Sheets*aufgrund der zahlreichen Verstöße eine ausführliche Überarbeitung des Konzepts erfordern.

Basierend auf den Erweiterungs- und Aktivierungsmöglichkeiten kann keine Entscheidung zwischen *Microsoft Excel* und *LibreOffice Calc*getroffen werden. Beide Anwendungen ermöglichen eine Erweiterung in mehreren Sprachen mit der Möglichkeit das entwickelte Modul systemweit zu integrieren. Existierende Bibliotheken zur Kommunikation mit der Ethereum Blockchain sind jeweils in mindestens einer Programmiersprache vorhanden. *Microsoft Excel* kann, aufgrund der Möglichkeit Code proaktiv auszuführen, ein marginaler Vorteil zugesprochen werden. Da das aktive Hören auf Änderungsereignisse allerdings auch reaktiv mittels Dokumentenevents wie *Öffnen* erreicht werden kann, ist dieser Vorteil vernachlässigbar. Auch die Unterschiede im Bereich der Auslösung und Behandlung von Änderungsereignissen sind nicht entscheidend. Da beide Anwendungen mehrere Möglichkeiten, Änderungsereignisse zu registrieren, bieten, sind die primären Unterschiede in der Anzahl der Operationen die Änderungsereignisse auslösen und in den enthaltenen Daten eines Änderungsereignisses zu finden. Während alle Änderungsereignisse in *Microsoft Excel* den Zugriff auf die geänderten Zellen teilweise einschließlich des vorigen Wertes zulassen, ist dies in *LibreOffice Calc* nicht für alle Methoden gegeben. Allerdings garantiert nur eine Implementierungsvariante in *LibreOffice Calc*, dass ein Änderungsereignis tatsächlich für jede Datenänderung auftritt. Um zu verhindern, dass jedes Dokument zwischengespeichert werden muss, sollte der Rückschluss eines Änderungsereignisses auf die geänderten Bereiche möglich sein. Da allerdings weder *Microsoft Excel* noch *LibreOffice Calc* dies zusammen mit einer garantierten Ereignisauslösung für jede Datenänderung bieten und auch der Umfang der fehlenden Ereignisse vergleichbar ist, sind beide Anwendungen ähnlich zu bewerten. *LibreOffice Calc* ist allerdings leicht besser zu bewerten, da der Programmfluss erweitert werden kann und es damit Operationen, die nativ kein Änderungsereignis auslösen, unterstützen kann. Letzteres ist auch der entscheidende Punkt in der verbleibenden Kategorie. Da in beiden Programmen sowohl Externalisierungsereignisse ausgelöst, als auch auf diese reagiert werden kann, besteht der primäre Unterschied darin, dass *LibreOffice Calc* es erlaubt ein Externalisierungsereignis abzubrechen, sollten Probleme bei einer Blockchaintransaktion auftreten. Damit kann garantiert werden, dass die lokalen Änderungen nicht unabhängig von der auf der Blockchain gehaltenen Version verändert werden. Zudem besteht in *Microsoft Excel* das Problem, dass aufgrund der Webumsetzung die Externalisierungsevents nicht garantiert identisch sind. Im Ganzen ist eine Umsetzung des erarbeiteten Konzepts in beiden Anwendungen möglich, mit einer leichten Tendenz zu *LibreOffice Calc*.

5.6 Zusammenfassung

Anschließend auf die Definition der Bewertungskriterien wurden drei Tabellenanwendungen untersucht. Die Ergebnisse dieser Untersuchung sind in Tabelle 5.1 zusammengefasst. Zwar existieren Änderungsereignisse in allen Programmen, jedoch sind diese auch alle in ihrer Vollständigkeit oder Nützlichkeit eingeschränkt. Insbesondere ist keiner der gefundenen Ansätze manipulationssicher. Zwar sind meistens Programmiervorkenntnisse für eine komplette Deaktivierung hilfreich, jedoch ist dies keine Einschränkung für Power-Nutzern von Tabellenprogrammen. Insbesondere fehlende

	Microsoft Excel		LibreOffice Calc		Google Spreadsheet	
	Makro	Add-In	Makro	Erweiterung	Add-On	Drive API
Verfügbare Änderungsereignisse	+	+	+	+	+	−
Kontrolle über Externalisierungsereignisse	∘	− −	+ +	+ +	− −	−
Leichtigkeit der Einbindung	+ +	+ +	∘	+ +	+	−
Aktivierungsaufwand	+ +	+ +	−	+	+ +	∘

Tabelle 5.1: Tabellenanwendungsvergleich

+ + nahezu keine Einschränkungen; bestes gefundene Ergebnis
+ geringer Umsetzungsaufwand, allerdings mit eingeschränkter Funktionalität
∘ erlaubt eine Umsetzung, die die gesetzten Ziele bereits nicht mehr voll erfüllt
− Umsetzung, die die Ziele nur unbefriedigend erfüllt; hat einen deutlich höheren Umsetzungswand
− − keine mögliche Umsetzung

Ereignisse bei normaler Verwendung des Programms ist bedenklich und stellt die hohen Sicherheitsansprüche des Konzepts in Frage. Die Online-Funktionalitäten von *Microsoft Excel* und *Google Sheets* entfernen Kontrolle über Externalisierungsereignisse. Allerdings auch unabhängig dieses Faktes bietet hierbei ausschließlich *LibreOffice Calc* volle Kontrolle. Installations- und Aktivierungsaufwand sind mit Ausnahme des Ansatzes über die *Google Drive API* und Makros in *LibreOffice Calc* vergleichbar. Insgesamt empfiehlt sich die Umsetzung als *LibreOffice Calc* Erweiterung. Die konkrete Umsetzung wird im nächsten Kapitel beschrieben.

6 Implementierung

Die in Kapitel 4 entwickelten Audit Trail Smart Contracts sollen nun mit dem in Kapitel 5 ausgewählten *LibreOffice Calc* verknüpft werden. Dieses Kapitel beschreibt zuerst die gewählte Integrations- und Aktivierungsmethode. Gefolgt von der Auswahl des speziellen Änderungslisteners und dessen Einschränkungen sowie dem Ablauf einer Transaktion. Abschließend folgt die Beschreibung der für die Visualisierung zuständigen Webkomponente.

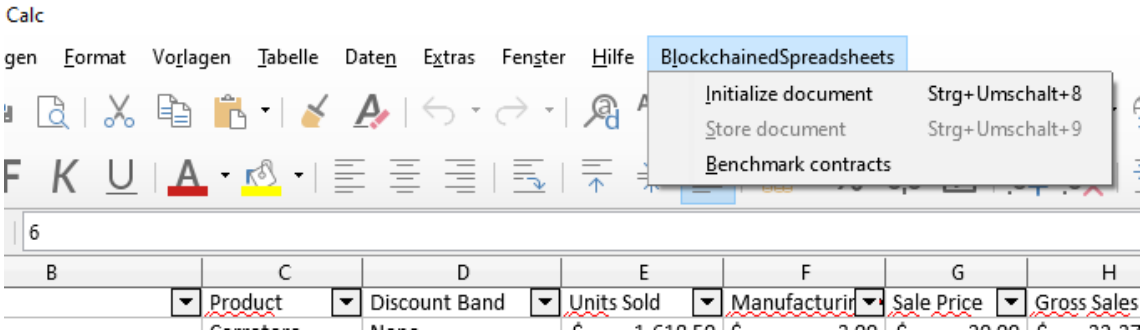
6.1 Integration in LibreOffice Calc

Sowohl die Benutzererfahrung, als auch Datenvollständigkeit sprechen gegen eine Integration über die händisch hinzugefügten Tabellenereignissen, welche in Abbildung 5.1b auf Seite 41 gezeigten wurden. Die Auswahl der Programmiersprache ist, gegeben der in Abschnitt 5.3 als nahezu identisch beschriebenen API, anhand des Ansatzes mit der am besten resultierenden Benutzererfahrung zu wählen. Eine installierbare Erweiterung ist der manuellen Markoverknüpfung zu bevorzugen. Konkret wird das Projekt in Java aufgrund der ausführlicheren Dokumentation auch in Voraussicht auf Ethereum-Transaktionen umgesetzt. Die Menüleiste wird um den in Abbildung 6.1a gezeigten Menüpunkt erweitert. Anhand des (de-)aktivierten Menüpunktes ist sofort erkennbar, ob Änderungen eines Dokumentes bereits verfolgt werden.

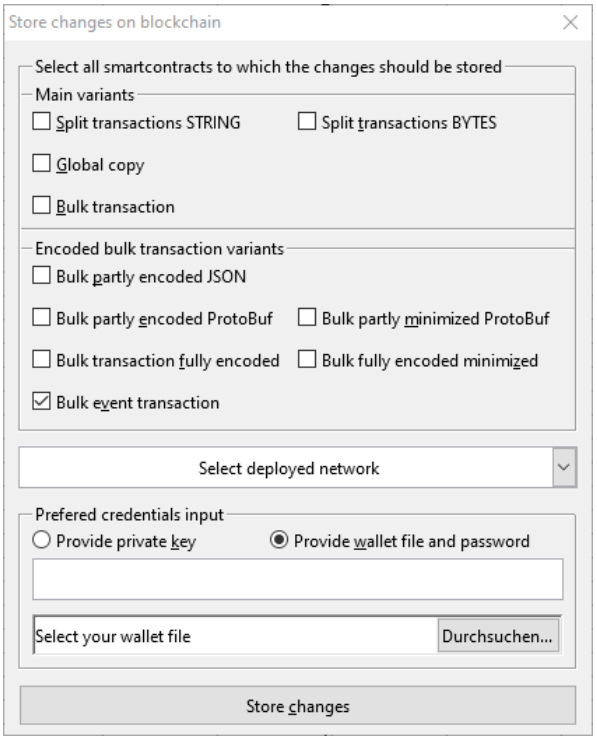
Wird das Audit Trail Modul für ein Dokument aktiviert, so werden alle aktuellen Werte zwischengespeichert und es wird um eine benutzerdefinierte Eigenschaft erweitert. Darin wird unter dem Schlüssel *blockchainId* der Dokumentidentifikator gespeichert. Dieser ist eine neu generierte UUID, wobei die Trennzeichen zwischen Gruppen entfernt wurden, das 32 Bytes Limit einer Solidity Variablen nicht zu überschreiten. Aufgrund des rein reaktiven Aufrufverhaltens von *LibreOffice Calc* wird für jedes neu geöffnete Dokument überprüft, ob diese Eigenschaft vorhanden ist. Ohne diesen Dokumentidentifikator werden Änderungen nicht beobachtet. Es können nur bereits lokal vorhandene Dokumente initialisiert werden um den Fokus als Änderungsprotokoll über Cloudspeicher oder Versionsverwaltung zu betonen. Auch würde eine gegenteilige Entscheidung die gewünschte Atomarität von Blockchain-Transaktion und lokalem Speichern noch weiter aufweichen. Ein Abbruch ist nach einer erfolgreichen priorisierten Blockchain-Transaktion nicht mehr möglich, jedoch könnte das Speichern danach noch abgebrochen werden.

6.2 Änderungsereignisse

Für die Auswahl zwischen beiden verfügbaren Änderungsereignissen ist eine genauere Betrachtung nötig. Die fehlende Zellbereichseinschränkung des *XModifyListener* würde bei einer Implementierung, das Zwischenspeichern aller gefüllten Zellen und deren vollständige Überprüfung erfordern.



(a) Eingefügte Menüleiste



(b) Dialog der *Blockchain-Speichern*-Operation

Abbildung 6.1: Grafische Elemente der LibreOffice Calc Erweiterung

Dies ist dadurch bedingt, dass ein Änderungsereignis nicht zwangsläufig genau eine geänderte Zelle bedeutet. Der zu überprüfende Bereich kann zwar durch granulärere Registrierung reduziert werden, jedoch auf Kosten der initialen Aktivierung. Auch die Anzahl der zwischenzuspeichernden Zellen kann hiermit nicht reduziert werden. Dies wäre nur mittels der Registrierung eines *XModifyListener* pro Zelle möglich. Jedoch ist dies für faktisch 2^{30} Zellen (2^{20} Zeilen * 2^{10} Spalten) pro Tabellenblatt performancetechnisch nicht vertretbar. In diesem Fall lösen Zeilen- und Spaltenmanipulationen zudem ein Ereignis in allen nachfolgenden Zellen aus. Eine minimale Darstellung dieser Operationen ist unabhängig des Registierungsortes des *XModifyListeners* nur unter erhöhtem Aufwand möglich. Da alle theoretisch betroffenen Zellen kontrolliert werden müssen, um in der Lage zu sein eine lokale Bereichsmanipulation auszuschließen. Zusätzlich lösen auch nicht daten-

verändernde Operation diesen Listener aus, womit diese ausführliche Überprüfung ohne praktisches Ergebnis durchgeführt werden würde. Weiterhin existiert ein Bug, wodurch eine Änderung dieses Ereignis zweimal auslösen kann.

Der *XChangesListener* erlaubt die Umgehung dieser Probleme durch eine Referenz auf den geänderten Bereich, wobei er allerdings nicht alle möglichen Operationen unterstützt. Die Bereiche sind jedoch nicht minimal, womit false positives, die eine Änderung von „leerer“ Zelle zu „leerer“ Zelle feststellen, entstehen können. So können unter anderem Pivottabellen, Datenmanipulationen mittels Makros bzw. der API, die Sortierfunktion, oder das Hinzufügen bzw. Löschen einzelner Zellen nicht bemerkt werden. Um ein korrektes Änderungsprotokoll zu garantieren, werden diese Funktionen mit *Dispatch Interceptoren* deaktiviert. Makros können nicht komplett deaktiviert werden, jedoch durch Erhöhen der Makrosicherheitsstufe nahezu unbenutzbar für Nutzer werden. *Ausschneiden* und *Einfügen* wird nur als eine Aktion behandelt, weshalb ersteres in *Kopieren* und *Löschen* aufgeteilt wird. Auch löst die *Rückgängig*-Operation nach Zeilen- oder Spaltenmanipulationen diesen Listener nicht aus. Um über diesen Weg ein fehlerhaftes Änderungsprotokoll zu vermeiden, wird nach einer dieser Operationen der Undo-Zwischenspeicher geleert und damit eine automatisierte Wiederherstellung eines früheren Zustandes verhindern.

Tabellenblattereignisse werden nicht vom *XChangesListener* erfasst und es gibt keine expliziten Ereignisse dafür. Um einen möglichst geringen Einfluss auf die Performance selbst von sehr ausführlichen Tabellen zu haben, wird auf eine Mischung beider Listener gesetzt. Der *XChangesListener* dient dem Behandeln von Änderungen innerhalb eines Tabellenblattes, wogegen mittels dem *XModifyListener* das Umbenennen oder Löschen eines Tabellenblattes erfasst wird.

Die Änderungen werden in einer Map, welche die Einfügereihenfolge beibehält, zwischengespeichert. Dabei ist die absolute Zellenbezeichnung, bestehend aus Tabellenblattname und Zellenname, beispielsweise „$Tabelle1.$O$33“, der Schlüssel und der aktuelle Zelleninhalt der verbundene Wert. Ausnahme sind strukturverändernde Operationen, dies umfasst Zeilen- und Spaltenmanipulationen und Löschen bzw. Umbenennen von Blättern, für welche diese Operation den Schlüssel darstellt und der betroffene Bereich als Wert vermerkt wird. Die Map ist dabei als Singleton umgesetzt, da die Interaktion mit einer Erweiterung in *LibreOffice Calc* diese unabhängig existierender Instanzen neu instanziiert.

6.3 Transaktion

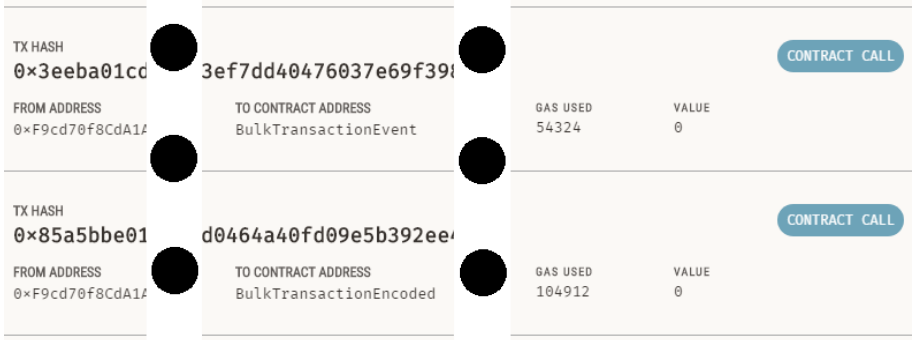
Die entwickelten Audit Trail Smart Contracts können mittels eines *Truffle*[1]-Scripts auf die Blockchain aufgespielt (deployed) werden. Unterstütze Testnetzwerke sind eine rein private Blockchain, der keine weiteren Clients hinzugefügt werden können, eine permissioned Blockchain, welche aus zwei Clients besteht und weiter erweitert werden kann und die öffentliche Blockchain *Rinkeby*[2]. Für die Kommunikation zwischen der Java Anwendung und der EVM wird Web3j[3] verwendet. Dabei kann der Nutzer nach einer der Externalisierungsoperationen *Drucken*, *Speichern* oder *Speichern unter* aus dem in Abbildung 6.1b dargestellten Dialog eine beliebige Anzahl an Smart Contracts und

[1] https://www.trufflesuite.com/truffle
[2] https://www.rinkeby.io/
[3] https://github.com/web3j/web3j

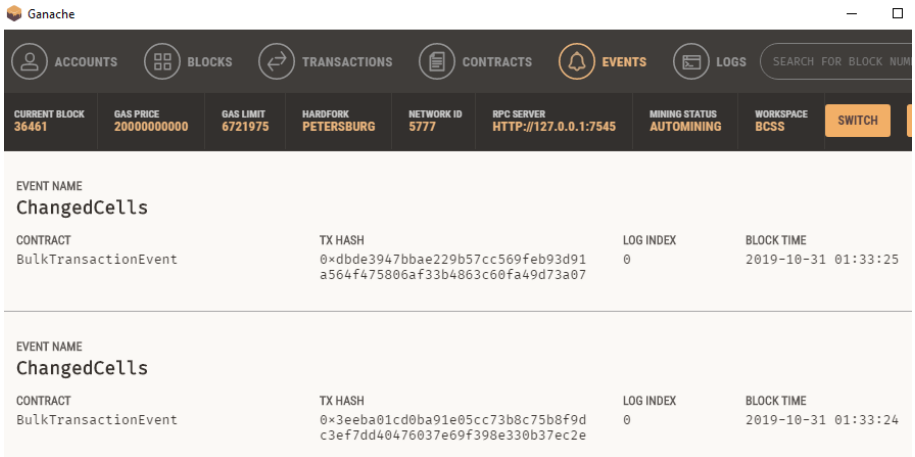
einen aktiven Ethereum Knoten auswählen. Nach Authentisierung mit einem Ethereum-Account werden alle zwischengespeicherten Änderungen in das geforderte Format umgewandelt und die Transaktionen ausgeführt. Dabei werden die ausgewählten Smart Contracts sequentiell ausgeführt, womit aufgrund der nicht instantanen Blockdauer, längere Wartezeiten entstehen können. Werden nicht alle Transaktionen angenommen, wird dies dem Nutzer als Fehlschlag des kompletten Vorgangs zurückgemeldet.

Abbildung 6.2a zeigt erfolgreich durchgeführte Transaktion der vorherigen Blöcke. In Abbildung 6.3 ist der World State der Grundvariante 3 nach einer abgeschlossenen Änderungstransaktion dargestellt. Die während Transaktionen ausgelösten Events werden in Abbildung 6.2b präsentiert.

Aufgrund der Begrenzung auf ein Zwischenspeicherungsformat wird auch an Variante 2 die kurzgefasste Beschreibung von Zeilen- und Spaltenmanipulationen übertragen. Da diese Variante nicht für dieses Format ausgelegt ist, kann dies in der Praxis zu fehlerhaften Spiegelungen auf der Blockchain führen. Für Variante 1 wurde die Annahme getroffen, dass es nur einen simultanen Nutzer gibt. Dies erlaubt es nur in der ersten Änderung den Dokumentidentifikator und Editor zu aktualisieren. In allen weiteren Transaktionen dieser Gruppe kann diese Information weggelassen werden und damit Kosten reduzieren. Können mehrere Nutzer gleichzeitig Änderungen speichern, so könnten Änderungen nicht mehr einfach dem verantwortlichen Editor und Dokument zugeordnet werden.



(a) Blockchain-Transaktionslog



(b) Blockchain-Event-Log

Abbildung 6.2: Audit Trail Modul Resultate

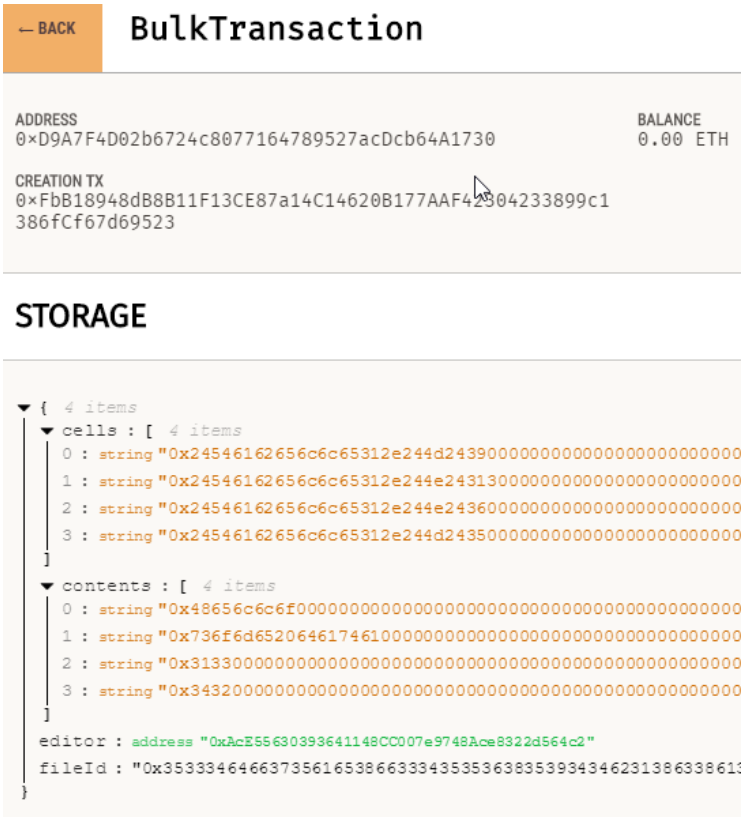


Abbildung 6.3: World State Repräsentation der unoptimierten Massentransaktion nach durchgeführten Änderungen

6.4 Visualisierung

Die Webanwendung zur Visualisierung des Audit Trails wird als ein auf Express[4] aufbauendes Node.js[5] Projekt umgesetzt. Es wurde entschieden die Visualisierung auf einen Smart Contract zu beschränken, da in einer realen Umgebung die Nutzung von mehreren unterschiedlichen Änderungsprotokollen als sehr unwahrscheinlich anzunehmen ist. Die Entscheidung fiel auf den voraussichtlich kostengünstigsten Smart Contract und damit die Event-Log Variante. Nach vergangenen Events wird mit Web3.js[6] auf dem lokalen Ethereum-Knoten gesucht. Diese werden decodiert und an Webclients übertragen. Abbildung 6.4 zeigt einen Teil der gerenderten Nutzeroberfläche. Ein Filterfeld dient zur Eingrenzung der dargestellten Änderungen nach Editor oder Dokumentidentifikator. Die drei definierten Metadaten identifizieren jede zusammengefasste Änderung. Diese ist dabei nach Tabellenblatt aufgeteilt und mit mehreren Veränderungen pro Zeile, notiert als *Zelle –> neuer Wert*, dargestellt. Änderungen der Zeilen- beziehungsweise Spaltenstruktur oder an ganzen Tabellenblättern werden farblich weiter hervorgehoben.

[4] https://expressjs.com/
[5] https://nodejs.org/en/
[6] https://github.com/ethereum/web3.js

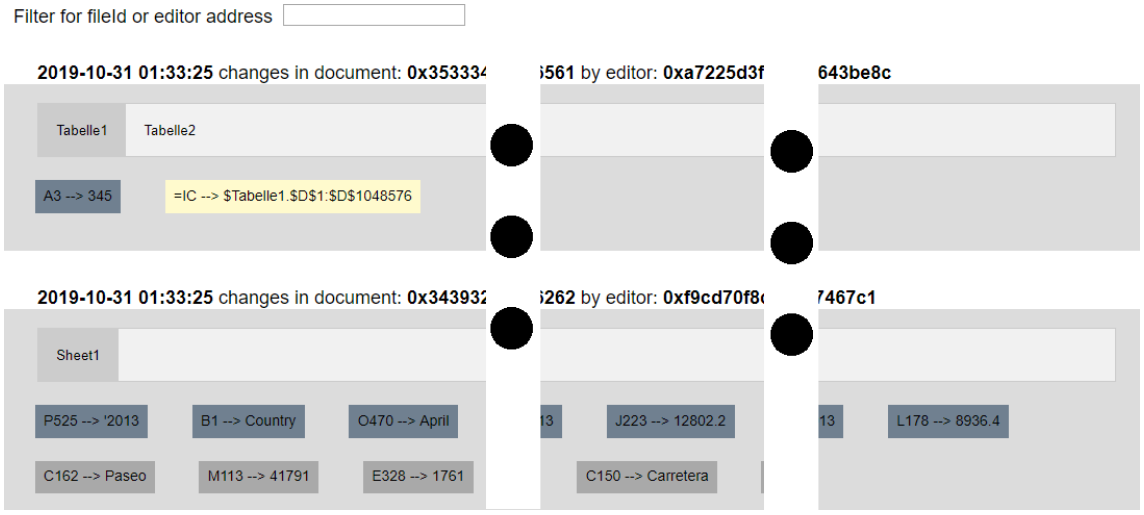


Abbildung 6.4: Weboberfläche des Audit Trails

6.5 Zusammenfassung

Die in Java durchgeführte Erweiterung von *LibreOffice Calc*sowie der Aufbau der Webkomponente wurden beschrieben. Das Zusammenspiel aller Komponenten und deren verwendete Technologien sind in Abbildung 6.5 dargestellt. Als Dokumentidentifikator wird eine UUID generiert, die den benutzerdefinierten Dokumenteigenschaften hinzugefügt wird. Änderungen des Tabellenmodels bzw. an den enthaltenen Daten werden mit dem *XChangesListener* erfasst und Veränderungen der vorhandenen Tabellenblätter werden durch den *XModifyListener* behandelt. Diese Änderungen werden zwischengespeichert bis der Nutzer eine der Externalisierungsoperationen startet. Darauf kann er aus den angebotenen Smart Contracts selektieren und muss Zugangsdaten zu seinen Ethereum-Account bereitstellen. Im Folgenden werden für die ausgewählten Smart Contracts Transaktionen mit den zu speichernden Daten sequentiell gestartet. Sind diese alle erfolgreich, wird die Externalisierungsoperation ausgeführt. Alle Änderungen des Eventlog Smart Contracts können visualisiert werden. Eine Exploration der entstehenden monetären Kosten wird nachfolgend ausgeführt.

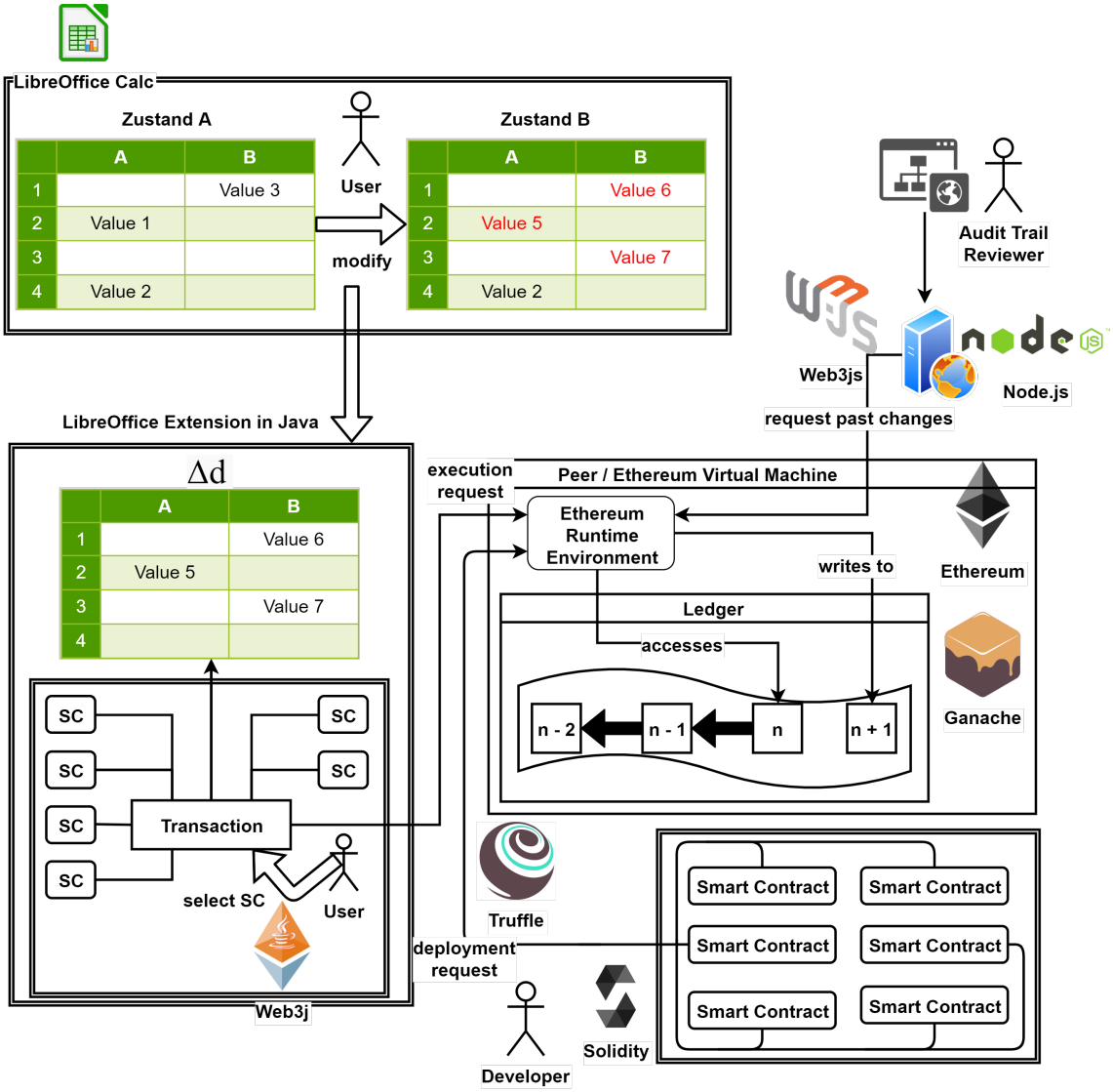


Abbildung 6.5: Technische Umsetzung des Konzepts

Änderungen in LibreOffice Calc durch eine in Java programmierte Erweiterung zwischengespeichert. Eine Externalisierungsoperation erfordert die Auswahl von bereitgestellten Smart Contracts. Diese kommunizieren über Web3j mit der EVM, die die Daten auf die Blockchain schreibt. Smart Contracts wurden in Solidity und können mit Truffle deployed werden. Ein Node.js Server erfragt mit Web3js vergangene Änderungen und rendert diese für die Darstellung auf einer Website.

7 Kostenabschätzung

Nachdem die vorgeschlagenen Änderungsprotokoll Smart Contracts mit einer Spreadsheet-Anwendung zu dem Audit Trail Modul kombiniert wurden, ist eine genauere Abschätzung der zu entstehenden Kosten angebracht. Dafür wird ein Laborexperiment durchgeführt. Der Aufbau und das Vorgehen werden beschrieben, um darauf die gemessenen Ergebnisse vorzustellen.

7.1 Vorgehen

Die mit einem in Kapitel 4 entwickelten Smart Contract Kosten können in einmalige und wiederkehrende Kosten aufgeteilt werden. Einmalige Kostenfaktoren ist lediglich die initiale Transaktion auf die Blockchain (Deployment). Transaktionskosten sind wiederkehrend und können in statische und dynamische Kosten aufgeteilt. Der Transaktionsbeginn als auch andere Operation, welche immer ausgeführt werden müssen, werden in die statischen Kosten eingerechnet. Dynamische Kosten sind dagegen von den zu transferierenden sowie den aktuell auf der Blockchain gehaltenen Daten abhängig. Letzteres trifft nicht auf die *gespaltenen Transaktionen* und *Ethereum-Event* Umsetzungen zu.

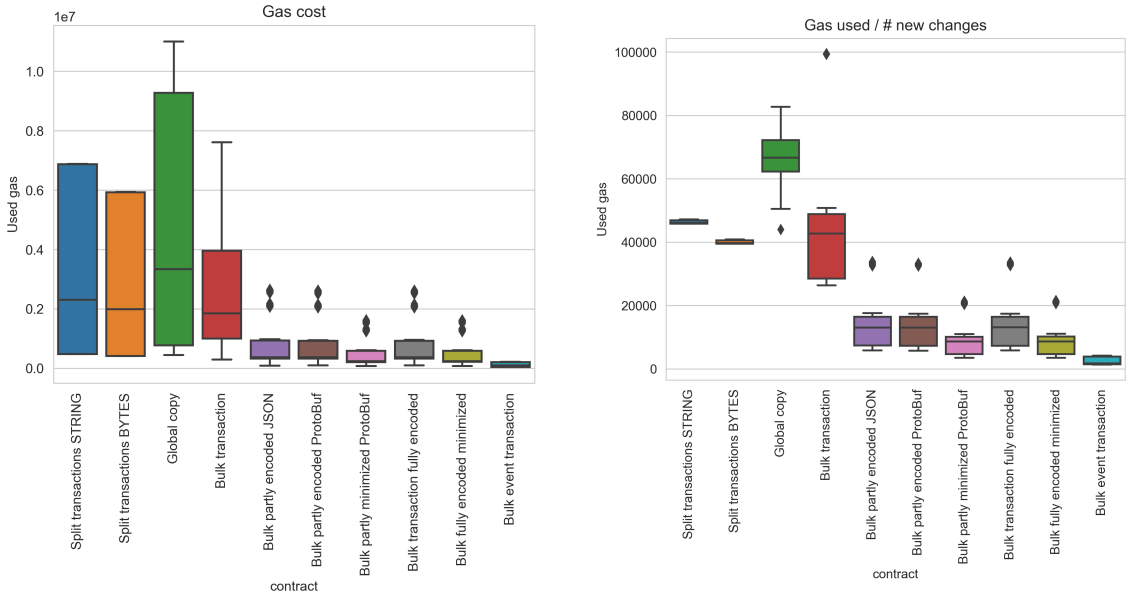
Als Datenbasis dient eine von Microsoft bereitgestellte Beispieldatei einer Finanztabelle[1]. Es werden drei Größenordnungen für sowohl die zu speichernden Daten als auch die bereits persistenten Daten in Form der direkt vorhergehenden Transaktion festgelegt. Die erste Gruppe repräsentiert kleine Änderungen und umfasst zehn geänderte Zellen. Eine Änderung mittlerer Größe entspricht 50 bearbeiteten Zellen. Umfassende Dokumentänderungen, dargestellt mittels 150 anderen Zellen, sind die letzte betrachtete Größenordnung.

Die empirische Kostenermittlung erfolgt für alle entwickelten Smart Contracts und für jede Kombination aus aktueller und vorhergehender Transaktionsgröße. Zwei unabhängige Änderungsmengen werden zufällig mit nicht leeren Werten der Beispieldatei befüllt. Für den ersten entwickelten Smart Contract wird der Ausgangszustand der ersten Größenordnung hergestellt, worauf die zu bewertende Transaktion in der kleinsten Größenordnung ausgeführt wird. Die für die Transaktion benötigten Gaskosten werden notiert und der Ausgangszustand wird wiederhergestellt und selbiges für die verbleibenden beiden Größenordnungen wiederholt. Nach selbigem Prinzip wird für die anderen Ausgangszustandsgrößenordnungen vorgegangen. Dieses Vorgehen wird für alle entwickelten Smart Contracts wiederholt. Dabei sind die verwendeten Zellen und ihre zugehörigen Werte in jeder Größenordnung identisch. Es werden sowohl die aufaddierte Länge von Zellenidentifikator und Zelleninhalt festgehalten. Dieser komplette Prozess wird 20 Mal wiederholt. Die Ethereum-Blockchain

[1] https://go.microsoft.com/fwlink/?LinkID=521962

Audit Trail Smart Contract	*Gas*-Kosten des Deployments	Monetärer Gegenwert in US$
Split transactions STRING	601 113	2,08
Split transactions BYTES	450 965	1,56
Globale Kopie	1 293 523	4,48
Masssentransaktion	441 223	1,53
Masssentransaktion teilweise encodiert	329 275	1,14
Masssentransaktion komplett encodiert	259 674	0,90
Masssentransaktion Ethereum-Event	179 905	0,62

Tabelle 7.1: *Gas*-Kosten und ihr Wert in US$ des Deployments



(a) Insgesamt benötigte *Gas*-Kosten in Wei

(b) *Gas*-Kosten in Wei pro neuer Änderung

Abbildung 7.1: Gemessene *Gas*-Kosten nach Smart Contract

wird mittels *Ganache*[2] bei einem Gaslimit von 67 219 750 pro Block, was dem zehnfachen der aktuellen Ethereum-Blockgröße entspricht, und unmittelbarer Verarbeitung von Transaktionen, simuliert. Um selbst praktisch zu große Transaktionen verarbeiten zu können wurde diese hohe Blockgröße gewählt. Damit kann garantiert werden, dass der stattfindende Vergleich auf repräsentative Daten aufbaut und nicht fehlende Daten eine Reduktion der Aussagekraft erwirken.

Audit Trail Smart Contract	Gesamte *Gas*-Kosten		*Gas*-Kosten pro gemachter Änderung		*Gas*-Kosten pro enthaltenem Byte	
	Durch-schnitt (Mean)	SD	Mean	SD	Mean	SD
Split transactions STRING	3 212 484	2 699 094	46 238	511	2 391	100
Split transactions BYTES	2 770 060	2 325 944	39 917	510	2 064	87
Globale Kopie	4 624 398	3 888 842	66 926	7 178	3 458	377
Masssentransaktion	2 808 174	2 493 521	45 245	21 224	2 332	1 075
Masssentransaktion teilweise encodiert als JSON	855 459	851 379	13 817	7 952	712	403
Masssentransaktion teilweise encodiert als ProtoBuf	846 079	841 097	13 673	7 843	704	398
Masssentransaktion teilweise encodiert als minimiertes ProtoBuf	526 762	510 064	8 827	4 885	455	248
Masssentransaktion komplett encodiert als ProtoBuf	849 452	840 511	13 814	7 867	712	99
Masssentransaktion komplett encodiert als minimiertes ProtoBuf	531 553	511 770	8 965	4 927	462	250
Masssentransaktion Ethereum-Event	111 761	70 808	2 370	1 144	123	60

Tabelle 7.2: Durchschnittliche *Gas*-Kosten aller Externalisierungsevents und pro enthaltener Änderung

7.2 Ergebnis

Die ersten entstandenen Kosten sind mit dem Deployment auf der Blockchain verbunden. Tabelle 7.1 listet diese einmaligen Kosten aller Smart Contracts auf. Variante 2 ist dabei, mit nahezu 1,3 Millionen *Gas*, der kostspieligste Smart Contract. Der String Ansatz von Variante 1 benötigte circa 600 000 Einheiten *Gas* und ist damit die zweitteuerste Umsetzung. Die Event-Variante weist mit

[2] https://www.trufflesuite.com/ganache

knapp 180 000 *Gas* die geringsten Kosten vor der komplett encodierten Massentransaktion (259 674 *Gas*) auf. Entstandene Kosten, basierend auf den in Abschnitt 2.2.1 genannten ETH und *Gas*-Preisen, sind schlimmstenfalls einstellig und bewegen sich zwischen ungefähr US$0,60 und US$4,50.

Tabelle 7.2 führt durchschnittliche *Gas*-Kosten, sowie eine Aufteilung in pro Änderung und pro Byte zusammen mit der jeweiligen Standardabweichungen, aller Umsetzungen auf. Ereignisse waren ungefähr ein Kilobyte (M = 1 354; SD = 1 139) groß, wovon jede Änderung circa 20 Byte (M = 19; SD = 0,8) in Anspruch nahm. Über die insgesamt 180 angestoßenen Externalisierungsereignisse pro Smart Contract hinweg, sind die benötigten *Gas*-Kosten eines Ereignisses in Abbildung 7.1a und pro gemachter Änderung in Abbildung 7.1b zusammengefasst. Untere bzw. obere Antenne bezeichnet den 1,5 fachen Interquartilabstand, außer Maximum bzw. Minimum liegen darunter. Der Median trennt das obere vom unteren Quartil. Auf alle Transaktionen bezogen sind Ausreißer nur in den teilweise und komplett encodierten Optimierungsvarianten aufgetreten. Der Ausreißer der nicht-optimierten Variante 3 bei einer Aufteilung in pro Änderung, bezieht sich auf den Fall einer kleinen Änderung (10) gefolgt auf eine große Änderung (150).

Die maximal mit einer einzigen Transaktion entstandenen *Gas*-Kosten von Variante 2 waren 10 998 523. Grundvariante 3 benötigte maximal 7 610 648 *Gas*. Die kombinierten Kosten von der String-Umsetzung von Variante 1 benötigte aufaddiert maximal 6 874 482 Einheiten *Gas*, während die Bytes-Umsetzung mit 5 926 458 auskam. Die Variante 4 kam mit 213 823 *Gas* aus, wogegen die komplett encodierte Umsetzung 1 613 612 *Gas* in der optimierten Implementierung, sowie 2 613 515 *Gas* für die nicht optimierte Umsetzung benötigten. Die JSON-Variante des teilweise encodieren Ansatzes benötigte im teuersten Fall 2 634 730 *Gas*. Für die unoptimierte ProtoBuf-Variante des gleichen Smart Contracts waren 2 611 005 *Gas* die Obergrenze. Als ein weiteres Box-Whisker-Plot präsentiert Abbildung 7.2 den Zusammenhang zwischen genutztem *Gas* und der Anzahl an enthaltenen Änderungen.

7.3 Zusammenfassung

Um die entstehenden Kosten des Audit Trail Moduls abschätzen zu können, wurde ein Laborexperiment durchgeführt. Änderungen wurden durch zufällig ausgewählte Werte einer Beispieldatei simuliert. Es existieren drei distinktive Größenordnungen und Transaktionen werden für jede Kombination aus alter und neuer Größenordnung ausgelöst. Die entstehenden Transaktionskosten werden zusammen mit diesen beiden Variablen notiert. Variante 1 und Grundvariante 3 verursachen ähnliche Kosten. Die teuerste Umsetzung ist Variante 2. Entstandene Kosten pro enthaltene Änderung konnten durch die Optimierungsvarianten von Variante 3 signifikant verringert werden. Die Event Umsetzung benötigt pro enthaltener Änderung unter 30% der verursachten Kosten der nächstgünstigsten Umsetzung.

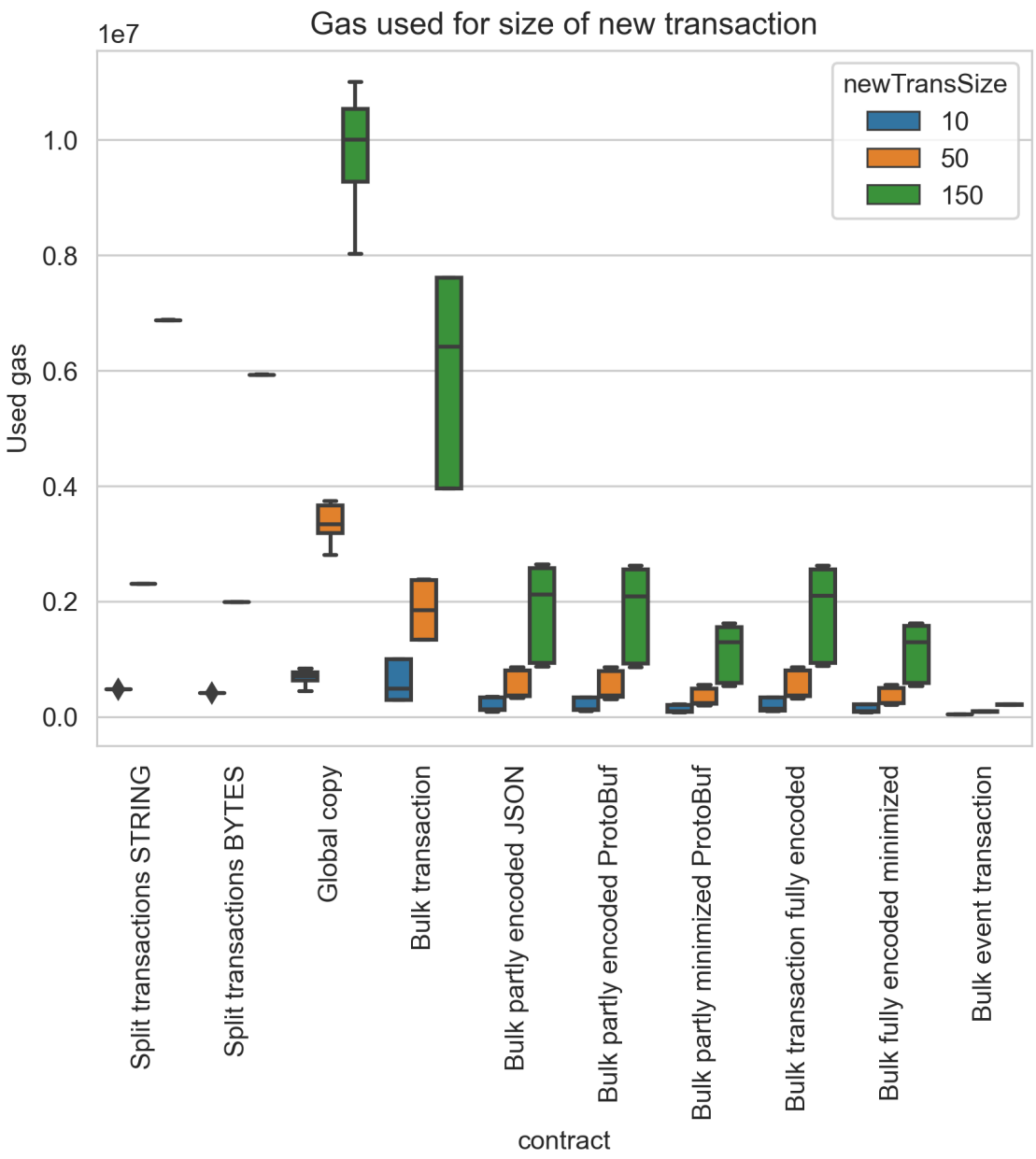


Abbildung 7.2: *Gas*-Kosten jeder Umsetzung nach Anzahl enthaltener Änderungen

Variante 4 hat durchweg die geringsten erwarteten Transaktionskosten. Die optimierten Ansätze von Variante 3 folgen darauf.

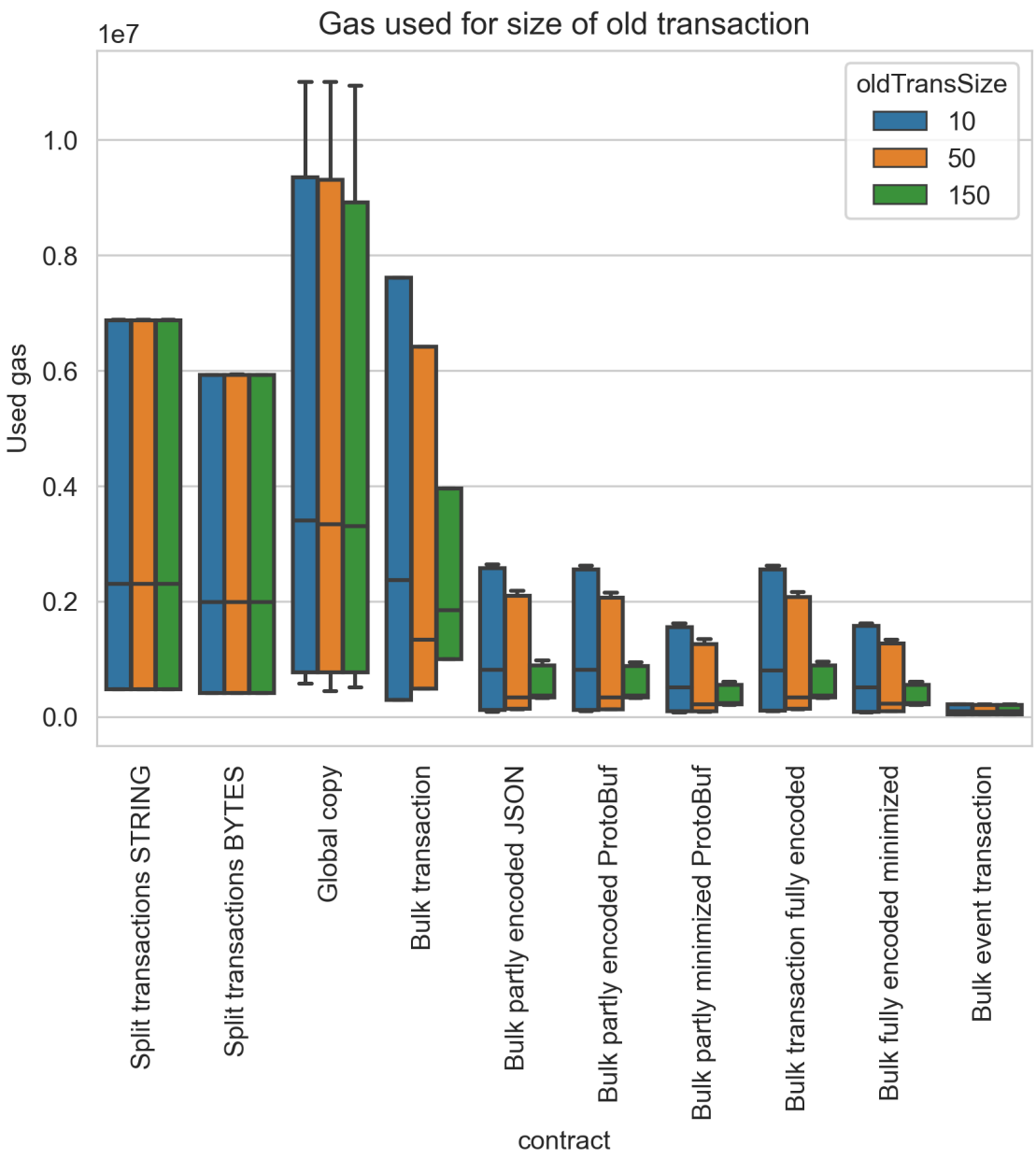


Abbildung 7.3: *Gas*-Kosten jeder Umsetzung nach Anzahl in der vorherigen Transaktion enthaltenen Änderungen

Variante 1 und 4 sind nicht von der Transaktionsgröße der vorherigen Transaktion betroffen. Für Variante 2 kann dies ebenfalls vermutet werden, jedoch enthält sie auch größten Schwankungen.

8 Diskussion

Wie bereits in Abbildung 7.1 und 7.2 zu sehen war, haben alle Umsetzungen, mit Ausnahme von Variante 1 und dem Event-Ansatz von Variante 3 eine hohe Spanne an Transaktionskosten. Dies kann entweder durch hohe statische Kosten einer Umsetzung oder durch Abhängigkeit von früheren Daten hervorgerufen werden. Im Folgenden werden sowohl die Umsetzungen anhand der gemessenen Werte als auch die Werte selbst diskutiert und in Relation zueinander gesetzt. Im Anschluss werden alle bekannten Einschränkungen kritisch betrachtet. Abschließend werden die gezogenen Schlussfolgerungen präsentiert.

8.1 Ergebnis

Obwohl die Deployment-Kosten von Variante 2 mehr als doppelt so hoch sind, wie die des nächstkostspieligsten Ansatzes, und mehr als sieben Mal höher im Vergleich zu der günstigsten Umsetzung, können diese Kosten im weiteren Verlauf vernachlässigt werden. Dies ist damit zu begründen, dass Datenspeicherungskosten die Deployment-Kosten aller Smart Contracts bereits vor der 100sten geänderten Zelle übersteigt. Jeder Smart Contract kann für mehrere Dokumente wiederverwendet werden, womit mit einer deutlichen Überschreitung dieser Grenze zu rechnen ist. Somit erscheint eine Fokussierung auf die durch Änderungen entstehenden Kosten als sinnvoll. Auch ist es nicht überraschend, dass die Initialisierung von Smart Contracts, welche kompliziertere Methoden oder mehr Variablen enthalten, in höheren Kosten resultiert. Es ist positiv hervorzuheben, dass die Einrichtung eines zentralen Datenspeichers für unter US$5 sehr attraktiv ist. Eine erstmalige Erstellung und Einrichtung einer vergleichbaren Lösung mit traditionellen IT-Lösungen würde, selbst ohne Garantie von Blockchain-Eigenschaften wie Manipulationsresistenz, Unternehmen ein Vielfaches kosten.

Basis des Kostenvergleichs mit einer traditionellen IT-Lösung ist ein Hybrid-Angebot von Amazons Amazon Web Services (AWS)[1]. Durch Nutzung des AWS Storage Gateway[2] werden die lokalen Daten in der Amazon S3[3] Cloud gespeichert. Dabei entstehen unter Nutzung des Datenzentrums Frankfurt, US$0,01 pro Gigabyte durch Nutzung des File Gateways. US$0,0245 an Kosten entstehen pro gespeichertem GB jeden Monat, angenommen die gespeicherten Daten überschreiten nicht 50 Terabyte. Damit sind die verbleibenden Kosten der traditionellen Lösung pro GB US$0,01 + (US$0,0245$*$ fortlaufender Monat). Da die größte Änderung unter Einbezug von Dokumentidentifikator und Editor-Account lediglich 3 033 Bytes betrug, kann davon ausgegangen werden, dass die entstehenden Änderungsdaten sehr klein bleiben. Dementsprechend ist nicht davon auszugehen, dass über ein GB an Daten in naher Zukunft entsteht. AWS berechnet den Preis pro angefangenen

[1] https://aws.amazon.com/
[2] https://aws.amazon.com/de/storagegateway
[3] https://aws.amazon.com/de/s3

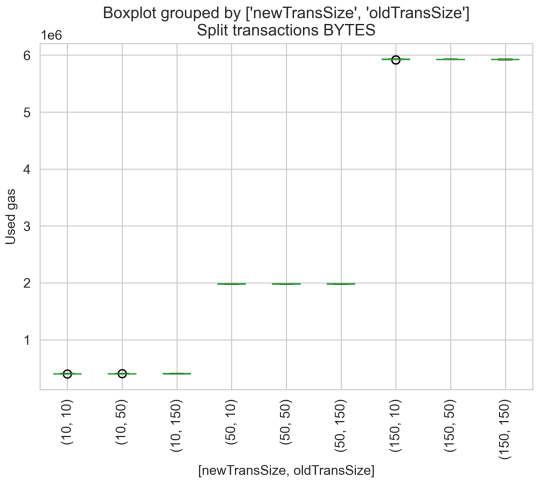
GB, das heißt, dass jeder Monat, in dem mindestens eine Tabellenänderung vorgenommen wird, mit US$0,0345 berechnet wird. Zusätzlich dazu kommen noch dynamische Kosten relational zu den benötigten Zugriffen. Gespeicherten Daten zu durchsuchen und zurückzugeben, wird mit US$0,00225 bzw. respektive US$0,0008 pro GB in Rechnung gestellt. Der Preis von weiteren Anfragen ist an die Anzahl ihres Auftretens gebunden. Pro tausend begonnene veränderte Aufrufe werden US$0,0054 berechnet, für durchsuchende Operationen wird US$0,00043 in Rechnung gestellt. Zusätzliche Kosten in Höhe von US$0,09 entstehen für jedes an das Internet übertragene, mit Ausnahme des ersten, Gigabytes. Auch hier ist davon auszugehen, dass diese Anwendungsfälle mindestens einmal pro Monat vorkommen. Der monatlich zu bezahlende Preis erhöht sich damit auf US$0,13338 pro Monat. Weil die gespeicherten Daten von der Blockchain kostenlos abgefragt werden können, müssen diese Kosten nicht für die Smart Contracts betrachtet werden. Potenzielle Gewinne durch erfolgreiches Mining von Blöcken werden nicht berücksichtigt.

8.1.1 Variante 1: Gespaltene Transaktionen (Split Transactions)

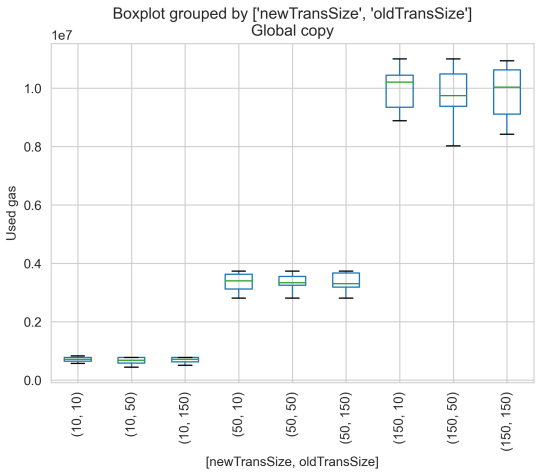
Wie anvisiert hat dieser Ansatz die geringste Kostenvarianz pro Änderung. Abbildung 8.1a zeigt, dass die Anzahl der vorherigen Änderung keinen Einfluss auf die entstehende Kosten haben. Dies ist damit zu begründen, dass der Transaktionsbeginn und andere statische Kosten nahezu die einzigen Kostenfaktoren darstellen. Bei einer optimierten Umsetzung als Event ist aufgrund des hohen Anteils der statischen Kosten keine Verbesserung auf ein der bestehenden Event-Umsetzung kompetitives Level zu erwarten. Auch ist wie in Abschnitt 4.1 angemerkt die vorherige Transaktionsgröße irrelevant. Aus Abbildung 8.2 kann der Vorteil einer Byte-Encodierung anstelle von dem intuitiven String-Äquivalent erkannt werden. So werden ungefähr 6 000 Einheiten *Gas* mehr pro enthaltener Änderung benötigt. Auch entsprechen die maximalen *Gas*-Kosten eines Ereignisses nahezu bzw. mehr als der aktuellen Blockgröße. Dies bedeutet, dass 150 Änderungen bereits effektiv einen komplett eigenen Block benötigten. Für die günstigere Bytes-Umsetzung würden pro Byte US$8,27 * e^{-3} berechnet Dies entspricht 8 272 040 pro GB. Jeder genutzter Monat des Vergleichssystems erlaubt die Speicherung von ungefähr 19 Bytes oder knapp einer Zellenänderung dieser Variante ohne eine Kostenerhöhung zu verursachen.

8.1.2 Variante 2: Globale Kopie (Global Copy)

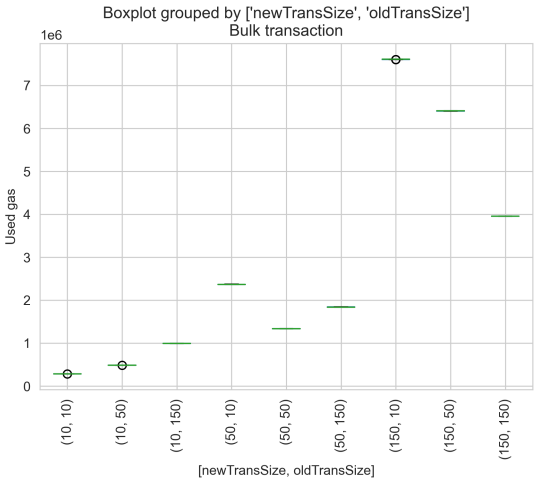
Auch dieser Ansatz resultiert in nahezu linearen Kosten unabhängig von der Anzahl der vorherigen Änderung, wie in Abbildung 7.1b und Abbildung 8.1b zu sehen ist. Die in der Konzeptionierung dieses Ansatzes getroffene Vorhersage der höchsten Kosten hat sich bestätigt. So treten in dieser für die mittlere und hohe Anzahl an Änderung die höchsten Kosten auf. Dies kann in Abbildung 7.2 nachvollzogen werden. Obwohl dies die einzige Variante ist, bei der eine längere Betriebsdauer zu geringeren Kosten führen kann, was aus der steigenden Varianz für große Änderungen in Abbildung 8.1b zu erkennen ist, sind die insgesamt entstehen Kosten nahezu nicht vertretbar. Die verbundenen *Gas*-Kosten sind dabei so hoch, dass selbst das normale Blocklimit nicht für die großen Transaktionen ausreicht. Jedes gespeicherte Zellenbyte kostet praktisch US$0,01197 womit sich das Persistieren von einem GB auf US$11 965 799 belaufen würde. Um gleichbleibende Kosten mit dem Vergleichssystem zu erreichen, dürften lediglich 11 Bytes pro Monat geändert werden. Dies entspricht einer Änderung jeden zweiten Monat.



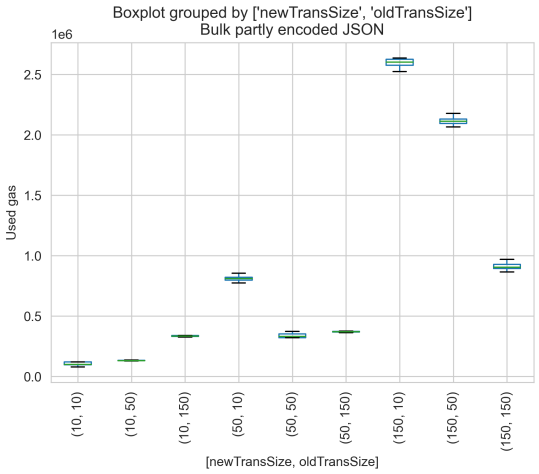
(a) Variante 1 Bytes Umsetzung



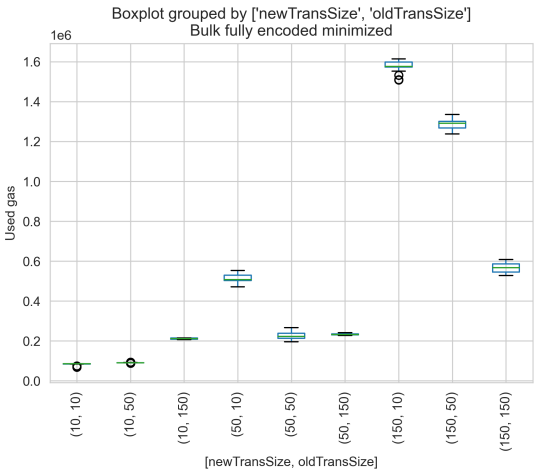
(b) Variante 2 Globale Kopie



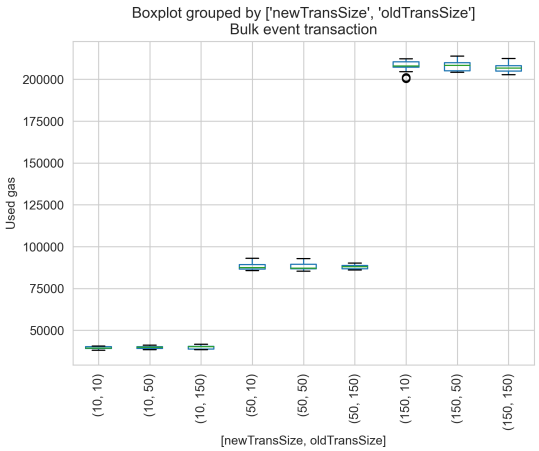
(c) Variante 3 Massentransaktion



(d) Variante 3 Optimierung teilweise encodiert als JSON

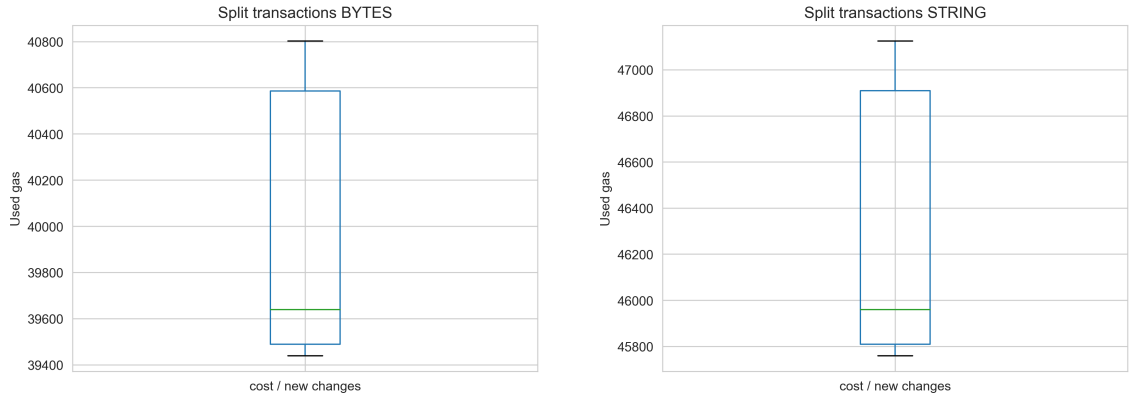


(e) Variante 3 Optimierung komplett encodiert als minimiertes ProtoBuf



(f) Variante 3 Optimierung Event Umsetzung

Abbildung 8.1: *Gas*-Kosten in Abhängigkeit der Anzahl neuer und alter Änderungen



(a) Bytes Umsetzung **(b)** String Umsetzung

Abbildung 8.2: *Gas*-Kosten in Wei pro neuer Änderung der Umsetzungen von Variante 1

8.1.3 Variante 3: Massentransaktion (Bulk Transaction)

Gegeben der abhängig von der Größe der zuvor im World State gehaltenen Daten weisen alle Umsetzung dieser Variante die höchste Varianz auf. Abbildungen 7.1b und 7.2 sowie Abbildung 8.1 veranschaulichen dies. Aus Tabelle 5.1 geht hervor, dass die durchgeführte Optimierung die entstehenden Kosten signifikant mit mindestens Faktor drei reduziert. Eine Encodierung als ProtoBuf über JSON verringert die Kosten minimal wogegen eine komplette Encodierung in einer neuerlichen minimalen Erhöhung der verbundenen Kosten resultiert. Das optimierte ProtoBuf-Schema kann die Kosten pro Byte um circa 35% pro ursprünglichem Byte im Vergleich zu den nicht optimierten Versionen verringern. Die unoptimierte Grundvariante erlaubt 17 Bytes, die minimierten ProtoBuf Umsetzungen 84,71 bzw. 83,42 Bytes für die teilweise encodierte Umsetzung respektive komplett encodierte Umsetzung pro Monat bei unveränderten Kosten. Der teilweise encodierte Optimierungsansatz mit minimierter ProtoBuf-Darstellung erreicht bereits US$1,57 * e^{-3}$ pro beigefügten Byte, wodurch die Kosten eines GB an Daten auf US$1 574 537 reduziert werden können.

8.1.4 Variante 4: Ethereum-Event (Bulk Event Transaction)

Der Event-Ansatz hat, wie in der Entwicklung vorgesehen, die geringsten Kosten verursacht. Dies gilt sowohl pro Änderung, als auch über alle Änderungen hinweg, wie in Abbildung 7.1 zu erkennen ist. Abbildung 8.1f bestätigt zudem, dass die Anzahl der vorherigen Änderungen keinen Effekt auf die entstehenden Kosten hat. Kosten steigen folglich proportional mit der Anzahl an Änderungen. In dieser Umsetzung kostet jede geänderte Byte nur noch US$4,24 * e^{-4}$, oder US$424 024 pro GB, womit pro fortlaufendem Monat 314,56 Bytes auf der Blockchain anstelle in der Vergleichslösung gespeichert werden können.

8.1.5 Vergleich

Variante 1 und die unoptimierte Variante 3 verursachen wie bei der Konzeptionierung gemutmaßt nahezu identische Kosten. Dies ist der Fall, obwohl Variante 1 dynamische Größen unterstützt, welche teurer sind. Bei einer Limitierung der unterstützten Änderungslänge und Zellenidentifikatorlänge auf 32 Byte wie in den anderen beiden Grundvarianten, könnten trotz durchschnittlich mehr benötigtem Speicherplatz die verbundenen *Gas*-Kosten weiter reduziert werden. Im Weiteren sollte von Nutzung der Grundvarianten 2 und 3 abgesehen werden solange dynamische Felderlängen das heißt verschachtelte Arrays in Solidity noch nicht sicher verwendet werden können. Allerdings hätte dies eine weitere Erhöhung der entstehenden Kosten zur Folge.

Als einziges Alleinstellungsmerkmal von Variante 1 verbleibt die Erstellung von Zwischenergebnissen. Diese Eigenschaft resultiert jedoch in hohen Kosten und langen Wartezeit. Dies ist ein nicht zu vernachlässigendes Argument gegen eine weitere Fortführung dieser Variante. Auch die Linearität der verursachten Kosten kann dies nicht ausgleichen, da sie auch mit der entwickelten Event-Umsetzung geteilt wird. Gegeben der geringeren Kosten dieser empfiehlt sich eine Abwendung von diesem Ansatz, falls die Speicherung der Daten nach dem Prinzip *alles oder nichts* kein direktes Ausschlussargument ist. Aufgrund der höheren Kosten läuft die Variante allerdings eher in Probleme als Variante 4.

Lediglich Variante 2 rechtfertigt die Nutzung des World States. Da durch die Überschreibung der aktuell gehaltenen Daten in den anderen beiden Varianten ältere Blöcke zur Hilfe genommen werden müssen, ist eine Umsetzung als Event deutlich kostengünstiger und sinnvoller. Dieser Ansatz entspricht effektiv einer sehr rudimentären Tabelle. Ein kommerzieller Erfolg ist sehr unwahrscheinlich gegeben dem insbesondere im Vergleich mit existierenden Tabellenprogrammen geringen Funktionsumfang. Eine dafür benötigte Erweiterung der angebotenen Funktionalität würde die bereits hohen Kosten von US$0,01 pro Änderung zudem weiter erhöhen. Im Übrigen müssten Nutzer gegeben der maximal erreichten Transaktionskosten zum regelmäßigen Speichern gezwungen werden um eine Annäherung oder gar die Überschreitung normale Blockgröße zu verhindern. Aufgrund der hohen entstehenden Kosten, welche bereits bei einer mittelgroßen Änderung die Kosten der großen Änderung aller optimierter Umsetzungen überschreitet, ist von einer weiteren Verfolgung dieses Ansatzes abzusehen.

Variante 4 liefert nicht nur das kosteneffizienteste Ergebnis, sondern erlaubt auch die schnellste Auflistung aller Änderungen. Zusätzliche Filter nach Dokumentidentifikator können hier bereits direkt umgesetzt werden. Ausschließlich Variante 2 erlaubt eine effizientere Wiedergabe des aktuellsten Stands. Auch die Linearität der Kosten, sowie Unabhängigkeit von der vorherigen Transaktionsgröße sind wünschenswerte Eigenschaften. Eine Fortführung dieser Umsetzung erscheint am sinnvollsten, jedoch sprechen die insgesamt noch immer sehr hohen initialen Kosten gegen eine Auslagerung aller Daten auf die Blockchain. In dem untersuchten Kontext mit sehr kleinen Files kann dies in Verbindung mit der Risikominimierungsstrategie eines Unternehmens bedarfsorientiert eine wertvolle Alternative sein. Riesige Datensätze die keine direkte finanzielle Entscheidungsrelevanz besitzen wie beispielsweise im ML vorhanden sollten dagegen tendenziell weiter gleich behandelt werden.

8.2 Einschränkungen

Gegeben der Differenzen zwischen den durchgeführten Größensamples sind die statistischen Werte über alle Transaktionen hinweg ohne eine Bezugnahme auf die verbundene Änderungsgröße nicht aussagekräftig. Dies spiegelt sich auch in der hohen Standardabweichung wider, wo diese unter anderem über 99,52% der durchschnittlichen Änderungsgröße beträgt. Durch den gegebenen Versuchsaufbau ist keine stetige Gleichverteilung der Ergebnisse zu erwarten. Aus Tabelle 8.1 welche Differenzen zwischen oberen Quartil und gemessenen Maximum sowie unterem Quartil und Minimum auflistet, ist erkennbar, dass drastische Veränderungen nach unten deutlich kleinere Auswirkungen haben als nach oben. Es ist zudem anzumerken, dass in der Realität höhere Kosten für sowohl für Variante 1 als auch Variante 2 zu erwarten sind. Für Variante 1 hängt dies mit der in der Implementierung getroffenen Annahme von lediglich einem gleichzeitigen Nutzer und damit gespartem Update von Dokumentidentifikator und Editor nach der ersten Transaktion. Durch Exklusion von Zeilen- und Spaltenänderungen als auch Umbenennung bzw. Löschung von Tabellenblättern in dem durchgeführten Benchmark wurden die im Vergleich zu den anderen Smart Contracts teureren Operationen von Variante 2 nicht betrachtet. Gegeben der ohnehin sehr hohen Kosten dieser Ansätze bestätigt dies allerdings eher die getroffene Entscheidung eine Weiterführung abzulehnen, anstatt diese infrage zu stellen.

Im Fall, dass Daten überproportional häufig wieder abgerufen werden sollte die Nutzung der Blockchain stärker in Betracht gezogen werden. Durch die kleinen untersuchten Dateien ist keine Überschreitung der vorhandenen Datenlimits zu erwarten. Deshalb könnten die erwarteten Kosten einer bereits anderweitig genutzten traditionellen Lösung gegen null gehen. Abschließend ist es auch noch schwer die Repräsentativität der genutzten Tabelle zu beurteilen. Diese hat zwar über 10 000 gefüllte Zellen, in denen sowohl Text als auch Zahlenwerte enthalten sind, jedoch ist es nur eine Beispieldatei, welche nicht zwangsläufig die reale Welt widerspiegelt oder auf alle Anwendungsbereiche von Spreadsheets angewendet werden kann. Die geringe Standardabweichung von 0,8 Byte pro Änderung erlaubt zumindest einen aussagekräftigen Vergleich der entstandenen Daten.

8.2.1 Ansatz

Die durchgeführte Kostenabschätzung hat eine weitere grundlegende Einschränkung, so gelten Kryptowährungen als nicht stabil und Schwankungen des ETH-Preises kommen regelmäßig vor. Dies hat offensichtlich eine Auswirkung auf die untersuchte Wirtschaftlichkeit. Auch wäre diese nicht gegeben, wenn die Kostenfunktion von der als Vergleichspunkt genutzten AWS Alternative ähnlich granulär wäre wie in Ethereum. Würden das Vergleichssystem auch auf Byte-Basis abgerechnet werden, so würde die Amortisation einer Änderung mehr als eine Millionen Jahre dauern. Diese Rechnung basiert bereits auf der als am besten befundenen Variante 4, allerdings bei einer Beschränkung auf die reinen Speicherkosten. Auch ergibt jede weitere von der Blockchain gegebene Garantie in Form von durchgeführten Operationen eine Erhöhung des verbundenen Preises. Um beispielsweise die Kongruenz eines Dokumentes über mehrere Nutzer zu gewährleisten sind weitere Überprüfungen nötig, da ansonsten das *verlorene Update* Problem auftreten kann.

Zwischen der gewährleisteten Manipulationssicherheit und der öffentlichen Einsehbarkeit der zu speichernden Daten muss abgewogen werden. Es sind nur alle Blockchain-Eigenschaften bei der Nutzung einer öffentlichen Blockchain garantiert, in diesem Fall sind alle Daten öffentlich einsehbar.

Audit Trail Smart Contract	*Gas*-Kosten: Q1 - Minimum	*Gas*-Kosten: Maximum - Q3
Split transactions STRING	1 728	6 912
Split transactions BYTES	1 728	6 912
Globale Kopie	330 848	1 726 576
Masssentransaktion	708 364	3 656 836
Masssentransaktion teilweise encodiert als JSON	241 499	1 708 428
Masssentransaktion teilweise encodiert als ProtoBuf	233 301,5	1 694 313
Masssentransaktion teilweise encodiert als minimiertes ProtoBuf	138 125	1 032 323,75
Masssentransaktion komplett encodiert als ProtoBuf	232 148	1 692 259
Masssentransaktion komplett encodiert als minimiertes ProtoBuf	138 656	1 027 301
Masssentransaktion Ethereum-Event	2 196	8 789

Tabelle 8.1: *Gas*-Differenz zwischen unterem bzw. oberen Quartil und gemessenen Minimum bzw. Maximum

Folglich sollten diese verschlüsselt werden. Die Verantwortung dafür muss an die Nutzer übertragen werden, jedoch ist es fraglich ob diese das erforderte Grundwissen aufweisen oder die benötigten Werkzeuge zur Verfügung gestellt bekommen. Wird der Entschlüsselungsmechanismus an externe Parteien weitergegeben besteht die Möglichkeit, dass diese permanenten Lesezugriff auf alle vergangen und neuen Änderungen erlangen. Eine zeitliche Begrenzung ist nicht möglich, da das Verschlüsselungsverfahren nicht nachträglich geändert werden kann und die Daten zeitlich und örtlich unabhängig von der Blockchain gelesen werden können. Eine solche Weitergabe erscheint unausweichlich, da sonst kein Audit Trail durchgeführt werden kann genauso wie der Nachweis, dass es sich bei den auf der Blockchain gehaltenen Daten tatsächlich um das Änderungsprotokoll einer gegebenen Tabelle handelt, nicht zu erbringen ist. Boshafte Veröffentlichungen eines Schlüssels würden das zugehörige Dokument der Öffentlichkeit frei zugänglich machen. Im Falle von kritischen Finanzdokumenten kann dies für Unternehmen fatal sein. Bei einer unvorsichtigen Verhaltensweise der Nutzer beispielsweise durch Verwendung des gleichen Schlüssels für mehrere Spreadsheets sind die Folgen gleich um ein vielfaches schlimmer. Dementsprechend wird ein zusätzliches Management-Tool zwischen Verschlüsselungsverfahren und Dokument benötigt. Im Weiteren ist auch eine Nutzerverwaltung nötig die Ethereum-Accounts an die Editoren verteilt und ausreichend ETH-Guthaben auf jedem Account garantiert. Allerdings besteht auch die Gefahr der Veruntreuung des bereitgestellten Guthabens durch Einzelpersonen.

8.2.2 Implementierung

Die Integration mittels der bereitgestellten Schnittstellen stellt eines der größeren Probleme des entwickelten Prototyps dar. Da die angebotene Funktionalität auch grundsätzlich dem Nutzer zugänglich ist, kann eine Manipulation nicht ausgeschlossen werden. So würde beispielsweise das

Entfernen der *blockchainId*-Eigenschaft die Validität des Änderungsprotokolls stark beeinträchtigen oder komplett aufheben. Auch ist die Einschränkung der erlaubten Funktionalität aufgrund fehlender Vollständigkeit von Änderungsereignissen nicht wünschenswert. In einer späteren Weiterentwicklung sollten Änderungsereignisse deshalb entweder zuverlässiger sein oder Änderungen sollten anhand des zugrundeliegenden Datenformats erkannt werden. Allerdings besteht, wie Adler und Nash [2] bereits anmerkten die Gefahr, dass Daten mittels eines anderen Tabellenprogramms oder in der Rohdatei verändert werden, sobald diese nicht durch einen zentralen Mechanismus vor anderweitigem Zugriff geschützt sind. Stehen wie im entwickelten Prototypen mehrere mögliche Smart Contracts als Speicherort zur Verfügung, sollte ein Verfahren entwickelt werden, welches den relevanten Smart Contract selbstständig erkennt und den Erfolgsstatus abhängig von diesem entscheidet. Auch könnte das sperrende Verhalten von Transaktionen und damit des Speichervorgangs von Nutzern als störend bewertet werden.

8.3 Folgerungen

Listing 8.1 Event Audit Trail mit unterstützter Versionierung und kontrollierten Editoren

```
contract SpreadsheetAuditTrail {
  struct File {
    bool isFile;
    address owner;
    address[] editorIds;
    mapping(address => Editor) editors;
    uint256 version;
  }
  struct Editor {
    bool isEditor;
    uint248 editorId;
  }

  bytes32[] internal fileIds;
  mapping(bytes32 => File) internal files;

  event ChangedCells(bytes32 indexed fileId, uint256 indexed version, bytes changes);

  function init(bytes32 _fileId) public returns (uint256) { ... }

  function addEditor(bytes32 _fileId, address _editor) public { ... }

  function removeEditor(bytes32 _fileId, address _editor) public { ... }

  function logNewestChanges(bytes32 _fileId, uint256 _version, bytes memory _changes)
        public returns (uint256) {
    require(files[_fileId].isFile);
    File storage file = files[_fileId];
    require(file.version == _version && file.editors[msg.sender].isEditor);
    _version = _version++;
    file.version = _version;
    emit ChangedCells(_fileId, _version, _changes);
    return file.version;
  }
```

Bei einer Fortsetzung des Konzepts sollte die Einführung von Änderungsberechtigungen und der Eliminierung von *verlorenen Updates* priorisiert werden. Eine solche Umsetzung ist in Listing 8.1 dargestellt. Jeder Account kann Dokumente initialisieren, jedoch kann nur dieser Account anderen die Berechtigung geben neue Änderungen an diesem Dokument in das Änderungsprotokoll einzutragen. Zudem werden Änderungen abgelehnt, falls diese nicht auf der aktuellsten Version durchgeführt wurden. Eine Zusammenführungsfunktionalität wäre mindestens auf Client-Seite wünschenswert. Durch eine kostenpflichtige Ablehnung der Änderungen, falls die oben genannten Bedingungen nicht erfüllt sind, wird zudem aktiv gegen Manipulation durch nicht berechtigte Accounts als auch Persistierung ohne Überprüfung nach einer neueren Version vorgegangen. Dieser Ansatz kostet

Listing 8.2 Fortgeführte Optimierung des Schemas aus Listing 4.5 auf Seite 35

```
message MapFieldEntry {
  int32 row = 1;
  int32 column = 2;
  string manipulation = 2;
  string value = 3;
}

```

892 760 *Gas* um auf die Blockchain aufgespielt zu werden (deployed). Das heißt die initialen Kosten haben sich im Vergleich zu Variante 4 nahezu verfünffacht. Eine *init* Operation kostet 106 415 *Gas*. Das Hinzufügen eines neuen Editors wird für den ersten mit 88 039 *Gas* und ab dem zweiten mit 73 039 *Gas* berechnet. Eine einzige Änderung der Größenordnung 10 durchgeführt vom Dokument-*Besitzer* kostete 41 730 *Gas*. Der Durchschnitt von Variante 4 in dieser Größenklasse betrug 39670 *Gas*. Dies zeigt, dass die Organisationskosten den Großteil der erhöhten Kosten ausmacht.

Das entworfene ProtoBuf-Format kann zudem weiter verbessert werden. Durch Aufteilen des Zellenidentifikators in zwei Integer-Werte, die die Position genauso beschreiben kann die teure String Encodierung umgangen werden. Das String-Feld wird für den Vermerk von Zeilen- bzw. Spaltenveränderungen sowie Tabellenblattmanipulationen benötigt. Eine Externalisierung der beiden Integer-Werte in eine eigene *Message* wäre zwar logisch sinnvoll jedoch weniger effizient.

8.4 Zusammenfassung

Obwohl keine Garantie gegeben werden kann, dass die gemessenen Daten für alle Anwendungsbereiche repräsentativ sind, besteht die Vergleichbarkeit untereinander. Von einer fortgesetzten Erforschung von allen vorgestellten Varianten mit Ausnahme von Variante 4 wird abgeraten. Diese ist nicht nur durchweg die günstigste Alternative, sondern erfüllt auch die in Kapitel 3 geforderte Erweiterbarkeit durch die integrierte Filterfunktion am besten. Allerdings sollte die Nutzung an den Anwendungszweck gebunden sein. Um nicht autorisierte Änderungen zu verhindern wurde Variante 4 um ein Nutzermanagement und Versionsnummern ergänzt. Die dadurch entstehenden Mehrkosten sind tendenziell im Organisatorischen anzusiedeln und es wird damit keine signifikante Auswirkung auf Änderungskosten erwartet.

9 Zusammenfassung und Ausblick

Tabellen sind in der Wirtschaft allgegenwärtig, enthalten allerdings häufig Fehler und sind damit ein Risikofaktor. Da sie insbesondere in dem finanziellen Entscheidungsprozess von Relevanz sind, können diese Fehler zu exorbitanten Kosten führen. Im Zuge des Risikomanagements sind Methoden vorgeschrieben, die Fehler verhindern als auch aufdecken sollen. Dazu gehören Audit Trails, mit denen gemachte Änderungen nachvollzogen werden können. Diese können jedoch selbst nachträglich manipuliert werden. Eine manipulationsresistente Technologie ist die Blockchain. Diese ist dezentralisiert und verwendet ein Konsensus-basierten Entscheidungsprozess der Mehrheit. Manipulationen können leicht erkannt werden, da die Erstellung eines korrekten Zustandes mit einem aufwendigen Algorithmus verbunden ist, wessen Ergebnis allerdings einfach zu verifizieren ist. Auf der Ethereum-Blockchain können eigenständige Programme ausgeführt werden, für die die Blockchain-Eigenschaften ebenfalls gelten. Dies ist jedoch mit Kosten verbunden. Die Datenspeicherung auf der Blockchain ist teil aktueller Forschungsarbeiten mit teils positiven Ergebnissen [30]. Eine Speicherung von Tabellenänderungsprotokollen auf der Blockchain könnte demnach ökonomisch sinnvoll sein.

Um dies zu untersuchen, wurde in Kapitel 3 ein Konzept eines mit der Blockchain kompatiblen Audit Trails ausgearbeitet. Dieses kann in die drei Grundaufgaben Tabellenanwendungserweiterung, Speicherung auf der Blockchain und Datenvalidierung aufgeteilt werden. Für erste und letztere Aufgabe sind off-chain Komponenten vorgesehen, während die zweite zwangsläufig on-chain umgesetzt werden muss. In der Tabellenanwendungserweiterung sollen Änderungsereignisse eines existierenden Tabellenprogramms genutzt werden, um Änderungen zu identifizieren und bis zu einem Externalisierungsereignis zwischenzuspeichern. Dieses löst eine Transaktion aus, welche diese Daten für die Persistierung auf der Blockchain vorbereitet. Eine öffentliche Blockchain ist anvisiert, jedoch ist die Umsetzung auf einer permissioned Blockchain während der Prototypphase auch ausreichend, da diese sich nicht in ihrer Funktionsweise unterscheiden, sondern lediglich die garantierten Eigenschaften in der später genannten reduziert sind. Die für das Änderungsprotokoll relevanten Daten beschränken sich auf eine Referenz zu dem geänderten Dokument, den verantwortlichen Editor und den Paaren aus Zellenidentifikatoren und neuen Zellwerten. Für eine ökonomische Speicherung dieser Daten ist die zweite Aufgabe verantwortlich. Im Zuge des Prototyps ist nur eine Visualisierungskomponente Teil der Datenvalidierung.

Da keine konkrete Speicherstruktur vorgeschrieben wurde, konnte diese Freiheit in Kapitel 4 genutzt werden, um mehrere Möglichkeiten zu erproben. Dabei entstanden vier Grundvarianten, welche sich durch die Anzahl aller gleichzeitig vorhandener Daten im World State unterscheiden. Die erste ist darauf beschränkt jederzeit nur die Daten einer Änderung sofort zugänglich bereitzustellen. In Variante 2 kann der komplette aktuelle Stand von Tabellen zurückgegeben werden. Alle Änderungen, die seit dem letzten Externalisierungsereignis entstanden sind, werden im World State der Variante 3 gespeichert. Zwei weitere Optimierungsvarianten, welche externe Serialisierungssprachen anstatt

von der Solidity-eigenen Speicherstruktur verwenden, wurden für diese Variante zusätzlich umgesetzt. In der letzten Variante werden die Daten nicht im World State gespeichert, sondern mittels Ethereum-Events in den Transaktionslogs hinterlegt. Die verwendete Struktur ist identisch mit der effizientesten Optimierungsvariante.

Microsoft Excel, *LibreOffice Calc* und *Google Sheets* wurden auf ihre Erweiterungsmöglichkeiten, den Aktivierungsaufwand von hinzugefügter Programmlogik, die Auslösung und Behandlung von Änderungsereignissen, sowie die Kontrolle über Externalisierungsoperation bewertet. Zwar sind Änderungsereignisse in allen untersuchten Anwendungen vorhanden, jedoch bietet keine einen manipulationssicheren Ansatz. Um Ereignisauslösungen komplett zu verhindern, sind meistens Programmierkenntnisse in einer der verbundenen Sprachen notwendig. Gegeben der langen Verwendungszeit von Tabellen, ist diese Einschränkung nicht als ausreichend zu bewerten. Auch bei einer normalen Nutzung lösen einige Operationen kein Ereignis aus. Damit genügt eine Erweiterung über die angebotenen Programmierschnittstellen jedoch nicht den erwarteten Sicherheitsansprüchen. In der Regel enthalten Ereignisse eine Referenz auf geänderte Zellen. Nur in *LibreOffice Calc* können Externalisierungsoperationen vollständige kontrolliert werden. Die beiden anderen Programme bieten entweder aufgrund ihrer Online-Funktionalität gar keine Informationen bezüglich dieser Aktionen an oder sind auf ein reaktionäres Verhalten beschränkt. Alle Anwendungen bieten ausreichende Einbindungsmöglichkeiten bei vertretbarem Aktivierungsaufwand. Im Gesamten hat *LibreOffice Calc* unter in Kapitel 5 verglichenen Tabellenprogrammen am besten abgeschnitten.

Nach Abschluss des Vergleichs wurde die konzeptionierte Tabellenanwendungserweiterung in *LibreOffice Calc* unter Verwendung der Java API durchgeführt. Für die Identifikation eines Dokuments wird eine UUID erstellt und in dessen benutzerdefinierten Eigenschaften gespeichert. Dokumente müssen nur einmalig initialisiert werden, worauf sie auch bei jeder folgenden Öffnung auf Änderungen beobachtet werden. Aufgrund fehlender Unterstützung mussten einige Funktionen allerdings deaktiviert werden. Vor Transaktionsbeginn kann aus allen angebotenen Smart Contracts und deren Datenencodierung ausgewählt werden. Nach Authentisierung mit einem Ethereum-Account werden die nötigen Transaktionen sequentiell angestoßen. Des Weiteren beschreibt Kapitel 6 die entwickelte Webkomponente zur Visualisierung aller vergangener Änderungen.

Aufbau und Durchführung des Laborexperiments sind in Kapitel 7 aufgeführt. Transaktionskosten der entwickelten Umsetzungen werden unter Betrachtung der enthaltenen Änderungen, sowie der Anzahl an Änderungen der vorherigen Transaktion gesammelt. Variante 2 hat sich als teuerste Umsetzung bestätigt. Dagegen ist Variante 4 durchweg mit den geringsten Kosten verbunden. Grundvariante 3 weist die größte Abhängigkeit von der vorherigen Transaktion auf und hat damit die höchste Standardabweichung pro enthaltener Änderung. Die Gesamtkosten dieses Ansatzes sind trotz dessen nahezu identisch mit Variante 1, welche die geringste Standardabweichung in dieser Kategorie aufweist. Die normale Blockgröße wurde von Variante 2 und Grundvariante 3 überschritten.

Die Diskussion in Kapitel 8 kommt zu dem Schluss, dass Variante 3 keinen Vorteil gegenüber Variante 4 bietet. Deshalb sollte von einer weiteren Exploration dieses Ansatzes abgesehen werden. Ähnliches gilt für die Varianten 1 und 2. Erstere besteht aus einem zu großen Anteil aus statischen Kosten und das einzige Alleinstellungsmerkmal sind die erstellten Zwischenergebnisse. Die Zweite bildet ein primitives Tabellenprogramm auf der Blockchain ab und internalisiert damit zusätzliche Funktionen. Aufgrund fehlender Funktionalität im Vergleich zu anderen Tabellenprogrammen ist eine solche Umsetzung unattraktiv. Im Allgemeinen haben sich beide Ansätze als zu teuer erwiesen, um eine zukünftige Verfolgung außerhalb der genannten Randfälle zu rechtfertigen. Auch Variante

4 ist teuer als das Vergleichssystem, jedoch könnte eine selektive Fortsetzung sinnvoll sein. Aus diesem Grund wurde dieser Ansatz abschließend um ein Autorisierungsprotokol erweitert. Durch dieses können nur ausgewählte Accounts Änderungen an einem Dokument durchführen. Und dies auch nur unter Bedingung, dass ihre Änderungen auf der aktuellsten Version ausgeführt wurden.

Ausblick

Die gesammelten Daten und Erkenntnisse sollten Grundlage einer zukünftigen Verbesserung der entwickelten Komponenten sein. Eine genaue Klassifizierung verschiedener Spreadsheet-Dokument anhand der enthaltenen Daten, ihrer Aufgabe und des verbundenen Risikos könnte durchgeführt werden. Die Risikoanalyse sollte überdies den für das Risikomanagement bereitgestellten Betrag enthalten. Diese Daten würden eine bessere Einschätzung liefern, ob die zusätzlichen Kosten annehmbar sind und können bereits in Design-Phase einer Tabelle als Entscheidungshilfe dienen. Eine genaue Evaluation des vorgeschlagenen Smart Contracts, bevorzugt mit diesen gesammelten Daten oder unter realen Bedingungen, sollte durchgeführt werden. Das Verhalten der Nutzer sollte in diesem Fall ebenfalls beobachtet werden. So könnten Rückschlüsse darauf gezogen werden, ob der gezwungene Arbeitsstopp bei einer Externalisierung negative Folgen hat. Würden entgegen der gegebenen Empfehlung mehrere Smart Contracts angeboten werden, sollte ein Verfahren entwickelt werden, mit dem die Zuordnung zwischen Dokument und Smart Contract automatisiert geschieht.

Von der Umsetzung als Erweiterung eines Tabellenprogramms ist in zukünftigen Ansätzen stark abzuraten. Sowohl die Funktionalität, auf welche zugegriffen werden kann, als auch die Gefahr von Nutzermanipulationen dieser können die hohen Sicherheitsanforderungen nicht erfüllen. Zwar könnten nachträgliche Änderungen verhindert werden, jedoch ist die Validität eines Änderungsprotokolls bei möglichen Verfälschungen vor dem Speicherzeitpunkt gefährdet. Diese Gefahr besteht jedoch immer, sobald das betroffene Dokument in der Arbeitsumgebung des Nutzers ist. Der Trend entwickelt sich ohnehin in Richtung von Webanwendungen, was diese Gefahr reduzieren könnte. Allerdings ist nicht intuitiv klar, was als Externalisierungsereignis in dem Cloud-Kontext gelten sollte. Auch ist das entwickelte Konzept nicht für mehrere gleichzeitige Editoren ausgelegt. Eine Fortführung der eigentlichen Datenvalidierung über die reine Visualisierung wäre auch angebracht.

Die Untersuchung der eingehaltenen Garantien und Kosten bei einer Speicherung der Daten off-chain in einer permissioned Blockchain und des Hashes on-chain wäre sinnvoll. Im Allgemeinen wäre eine genauere Betrachtung der Verwendung einer permissioned Blockchain in dem untersuchten Anwendungsfall interessant. Denn es kann argumentiert werden, dass obwohl diese Blockchain für Außenstehende nicht vertrauenswürdig ist die Nutzer der Tabellenanwendungen allein keine Mehrheit darstellen. Somit müsste sich die Mehrheit der Mitarbeiter für Manipulationen zusammenschließen. Diese Hürde könnte bereits abschreckend genug wirken.

Literaturverzeichnis

[1] URL: https://ethereum.stackexchange.com/a/65489 (zitiert auf S. 23).

[2] A. Adler, J. Nash. „Knowing what was done: uses of a spreadsheet log file“. In: *Spreadsheets in Education (eJSiE)* 1 (Okt. 2005) (zitiert auf S. 11, 14, 15, 68).

[3] S. Ammous. „Blockchain Technology: What Is It Good For?“ In: *Banking & Finance Law Review* 34.2 (2019), S. 239–251 (zitiert auf S. 16, 17).

[4] D. Birch, D. Lyford-Smith, Y. Guo. „The Future of Spreadsheets in the Big Data Era“. In: *Proceedings of the EuSpRIG 2017 Conference SSpreadsheet Risk Management", Imperial College, London, pp1-13 ISBN: 978-1-905404-54-4* abs/1801.10231 (2018). arXiv: 1801.10231. URL: http://arxiv.org/abs/1801.10231 (zitiert auf S. 13, 15).

[5] V. Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform*. (visited on 2019-10-25). URL: https://github.com/ethereum/wiki/wiki/White-Paper (besucht am 25. 10. 2019) (zitiert auf S. 16, 17, 19, 20, 23).

[6] V. Buterin. *Hard Fork Completed*. 20. Juli 2016. URL: https://blog.ethereum.org/2016/07/20/hard-fork-completed/ (zitiert auf S. 17).

[7] V. Buterin. *On Public and Private Blockchains*. 7. Aug. 2015. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ (besucht am 27. 10. 2019) (zitiert auf S. 16).

[8] T. Chen, X. Li, X. Luo, X. Zhang. „Under-optimized smart contracts devour your money“. In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Feb. 2017, S. 442–446. DOI: 10.1109/SANER.2017.7884650 (zitiert auf S. 19).

[9] T. Chen, Z. Li, H. Zhou, J. Chen, X. Luo, X. Li, X. Zhang. „Towards Saving Money in Using Smart Contracts“. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. Mai 2018, S. 81–84 (zitiert auf S. 19, 23).

[10] D. Colver. „Spreadsheet good practice: is there any such thing?“ In: *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2004 ISBN 1 902724 94 1*, arXiv:1001.3967 (Jan. 2010). arXiv: 1001.3967 [cs.HC] (zitiert auf S. 14).

[11] G. J. Croll. „Spreadsheets and the Financial Collapse“. In: *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2009 145-161 ISBN 978-1-905617-89-0* (2009). arXiv: 0908.4420 [cs.CY] (zitiert auf S. 11).

[12] A. DasGupta. „The matching, birthday and the strong birthday problem: a contemporary review“. In: *Journal of Statistical Planning and Inference* 130.1 (2005). Herman Chernoff: Eightieth Birthday Felicitation Volume, S. 377–389. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2003.11.015. URL: http://www.sciencedirect.com/science/article/pii/S0378375804002721 (zitiert auf S. 27).

[13] *Ethereum Gas Price History*. 30. Okt. 2019. URL: https://etherscan.io/chart/gasprice (zitiert auf S. 19).

[14] M. A. Ferreira, J. Visser. „Governance of Spreadsheets through Spreadsheet Change Reviews“. In: *CoRR* abs/1211.7100 (2012). arXiv: 1211.7100. URL: http://arxiv.org/abs/1211.7100 (zitiert auf S. 13–15).

[15] T. Grossman, V. Mehrotra, O. Ozluk. „Lessons from Mission-Critical Spreadsheets“. In: *Communications of the Association for Information Systems* 20.1 (Jan. 2007), S. 60. DOI: 10.17705/1CAIS.02060 (zitiert auf S. 11).

[16] V. Lemieux. „Archiving: The Overlooked Spreadsheet Risk“. In: *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2005 213-226 ISBN:1-902724-16-X* (Apr. 2008). arXiv: 0803.3231 (zitiert auf S. 14).

[17] T. Lemon, E. Ferguson. „A Practical Approach to Managing Spreadsheet Risk in a Global Business“. In: *Proc. European Spreadsheet Risks Int. Grp.* arXiv:1009.1404 (Sep. 2010), 15–22. 8 p. arXiv: 1009.1404 [cs.SE] (zitiert auf S. 14).

[18] L. Leon, L. Kalbers, N. Coster, D. Abraham. „A Spreadsheet Life Cycle Analysis and the Impact of Sarbanes-Oxley“. In: *Decis. Support Syst.* 54.1 (Dez. 2012), S. 452–460. ISSN: 0167-9236. DOI: 10.1016/j.dss.2012.06.006. URL: http://dx.doi.org/10.1016/j.dss.2012.06.006 (zitiert auf S. 13).

[19] A. G. Martin. „Get spreadsheets under control: without careful scrutiny, spreadsheets can contribute to financial reporting misstatements“. In: *Internal Auditor* 62.6 (2005), S. 31–35 (zitiert auf S. 11, 13, 15).

[20] J. Mattila. „The blockchain phenomenon“. In: *Berkeley Roundtable of the International Economy* (2016) (zitiert auf S. 16).

[21] T. Minter, C. Correia. „The governance of risk arising from the use of spreadsheets in organisations“. In: *Risk Governance and Control: Financial Markets and Institutions* 4 (Juni 2014), S. 7–15. DOI: 10.22495/rgcv4i2art1 (zitiert auf S. 13, 14).

[22] M. Montecchi, K. Plangger, M. Etter. „It's real, trust me! Establishing supply chain provenance using blockchain“. In: *Business Horizons* 62.3 (2019), S. 283–293. ISSN: 0007-6813. DOI: https://doi.org/10.1016/j.bushor.2019.01.008. URL: http://www.sciencedirect.com/science/article/pii/S0007681319300084 (zitiert auf S. 23).

[23] J. C. Nash, N. Smith, A. Adler. „Audit and Change Analysis of Spreadsheets“. In: abs/0807.3168 (2008). arXiv: 0807.3168. URL: http://arxiv.org/abs/0807.3168 (zitiert auf S. 11, 14, 15, 27).

[24] C. Noyes. „BitAV: Fast Anti-Malware by Distributed Blockchain Consensus and Feedforward Scanning“. In: *CoRR* abs/1601.01405 (2016). arXiv: 1601.01405. URL: http://arxiv.org/abs/1601.01405 (zitiert auf S. 23).

[25] R. Panko. „What We Don't Know About Spreadsheet Errors Today: The Facts, Why We Don't Believe Them, and What We Need to Do“. In: *Proc. 16th EuSpRIG Conf. SSpreadsheet Risk Management"(2015) pp79-93 ISBN: 978-1-905404-52-0* (2016). URL: https://arxiv.org/abs/1602.02601 (zitiert auf S. 13).

[26] R. R. Panko. „Spreadsheet Errors: What We Know. What We Think We Can Do“. In: *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2000 7-17 ISBN:1 86166 158 4* arXiv:0802.3457 (Feb. 2008). Comments: 9 Pages, 2 Tables. URL: http://cds.cern.ch/record/1090957 (zitiert auf S. 13).

[27] R. R. Panko. „What We Know About Spreadsheet Errors“. In: *Journal of End User Computing* 10.2 (Mai 1998), S. 15–21. ISSN: 1063-2239. URL: http://dl.acm.org/citation.cfm?id=287893.287899 (zitiert auf S. 13).

[28] M. Pilkington. „11 Blockchain technology: principles and applications“. In: *Research handbook on digital transformations* 225 (2016) (zitiert auf S. 17).

[29] S. G. Powell, K. R. Baker, B. Lawson. „A critical review of the literature on spreadsheet errors“. In: *Decision Support Systems* 46.1 (2008), S. 128–138. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2008.06.001. URL: http://www.sciencedirect.com/science/article/pii/S0167923608001127 (zitiert auf S. 11, 13).

[30] A. Ramachandran, M. Kantarcioglu. „Using Blockchain and smart contracts for secure data provenance management“. In: *CoRR* abs/1709.10000 (2017). arXiv: 1709.10000. URL: http://arxiv.org/abs/1709.10000 (zitiert auf S. 11, 23, 71).

[31] P. Sarda, M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han. „Blockchain for Fraud Prevention: A Work-History Fraud Prevention System“. In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. Aug. 2018, S. 1858–1863. DOI: 10.1109/TrustCom/BigDataSE.2018.00281 (zitiert auf S. 23).

[32] M. Sharples, J. Domingue. „The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward“. In: *Adaptive and Adaptable Learning*. Hrsg. von K. Verbert, M. Sharples, T. Klobučar. Cham: Springer International Publishing, 2016, S. 490–496. ISBN: 978-3-319-45153-4 (zitiert auf S. 23).

[33] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledge*. (visited on 2019-10-25). URL: https://ethereum.github.io/yellowpaper/paper.pdf (besucht am 25. 10. 2019) (zitiert auf S. 21–23).

[34] Y. Zhang, J. Wen. „An IoT electric business model based on the protocol of bitcoin“. In: *2015 18th International Conference on Intelligence in Next Generation Networks*. Feb. 2015, S. 184–191. DOI: 10.1109/ICIN.2015.7073830 (zitiert auf S. 23).

[35] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang. „An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends“. In: *2017 IEEE International Congress on Big Data (BigData Congress)*. Juni 2017, S. 557–564. DOI: 10.1109/BigDataCongress.2017.85 (zitiert auf S. 16, 17, 23).

Alle URLs wurden zuletzt am 11. 11. 2019 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift